Learning Online-Aware Representations using Neural Networks

by

Khurram Javed

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Learning online is essential for an agent to perform well in an ever-changing world. An agent has to learn online not only out of necessity — a non-stationary world might render past learning useless — but also because continual tracking in a temporally coherent world can result in better performance than a fixed solution. Despite the necessity of online learning, we have made little progress towards building robust online learning methods. More specifically, a scalable online representation learning method for neural network function approximators has remained elusive. In this thesis, I investigate the reasons behind this lack of progress. I propose the idea of online-aware representations – data representations explicitly optimized for online learning – and argue that existing representation learning methods do not learn such representations. I investigate if neural networks are capable of learning these representations. My results suggest that neural networks can indeed learn representations that are highly effective for online learning, but learning these representations online using gradient-based methods is challenging. More specifically, long-term credit assignment using back-propagation through time (BPTT) does not scale with the size of the problem. To address this, I propose Learning with Backtracking for slowly and continually improving representations online. The primary idea behind LwB is that while it is not possible to compute an accurate estimate of the representation update online, it is possible to verify if an update is useful online.

Preface

Chapter 4 of this thesis is based on a NeurIPS paper I co-authored with Martha White (Javed and White 2019) whereas preliminary work on Learning with Backtracking — introduced in Chapter 6 — was done in a paper co-authored with Martha White and Yoshua Bengio (Javed et al. 2020).

To Mom and Dad

For always supporting me in pursuing my goals and letting me make seemingly eccentric decisions.

It is all a matter of time scale. An event that would be unthinkable in a hundred years may be inevitable in a hundred million.

– Carl Sagan.

Acknowledgements

The ideas presented in the thesis have evolved over the past two years and would not have been possible without my interactions with numerous people in various capacities. First and foremost, I thank my advisor Martha White who has not only guided me in my research, but also made me a clearer thinker, a better writer, and a well-rounded researcher. Martha can cut to the essence of an idea quickly, which benefited me immensely in refining my ideas. I'm also grateful to her for providing me the freedom to pursue directions that excited me. Second, I'm grateful to Rich Sutton for instilling in me a drive to work on important problems. Rich's unwavering commitment to doing good science with a clear goal in mind has inspired me to aspire the same and has made my research more fulfilling. Third, I'm grateful to Adam White for making me a better critic of my as well as other's work; reviewing papers with Adam has had a profound impact on how I approach my research. Finally, I'm thankful to Yoshua Bengio for advising me during my visit to MILA. Yoshua's ideas have guided my research years even before I started my masters, and working closely with him has only increased his influence on my work.

I'm also thankful to my peers, colleagues, and friends at RLAI and Amii for providing a stimulating and fun environment for doing research. In particular, I would like to thank Abhishek Naik, Ehsan Imani, Fernando Hernandez, Han Wang, Kenny Young, Matthew Schlegel, Paritosh Goyal, Raksha Kumaraswamy, Roshan Shariff, Ryan Dorazio, Samuel Sokota, Shibhansh Dohare, Tian Tian, Vincent Liu, Yi Wan, and Zaheer Abbas. Raksha specifically made the two years more fun and the never-ending winters more bearable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

An agent interacting in the world gets constant feedback from the environment. This feedback can either come from passively observing the environment dynamics over time, or taking actions in the environment and observing the consequences of these actions. An agent that uses this feedback to adapt its behavior in real-time is broadly defined as an *online-learning system*. An *offline-learned system*, on the other hand, aims to learn a fixed solution for achieving goals and ignores the real-time feedback from the environment for adapting its behavior. An online learner has two distinct advantages over an offline-learned system.

First, an online learner can track — adapt to do well in the current part of the world at the expense of temporally distant parts. In a temporally coherent world, tracking can achieve better performance than a fixed solution (Sutton et al. 2007). This is because if the world is much larger than the agent — as often is the case — the agent might not have the capacity to represent the globally optimal solution. In such a setting, an online learner can focus its limited capacity to specialize on the part of the world it is currently in at the expense of temporally distant parts as shown in Figure 1.1. An offline learned system, on the other hand, has no option but to settle on a fixed sub-optimal solution.

Tracking is also essential if the world is non-stationary: no amount of offline learning can be sufficient in a world that changes (Silver et al. 2008); these changes can be due to changes in the environment — evolving seasons,

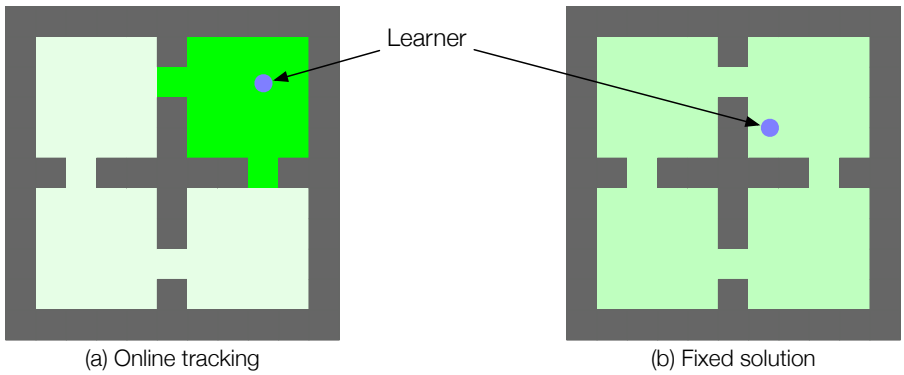


Figure 1.1: Online tracking vs a fixed solution in a four-room grid-world. The blue dot represents the position of the learner, and the intensity of the color — higher is better — of a room represents the performance of the learner in that room. I assume that the learner cannot represent the globally optimal solution due to limited capacity. In such a setting, an online learner can adapt its capacity to the room it is currently in at the expense of other rooms whereas an offline learner has to optimize for a solution valid for all rooms simultaneously. Assuming the learner spends more time in a room than it takes to adapt to a room change, the online learner can perform better than a fixed solution.

unpredictable natural disasters — changes in other agents — evolving prey-predator relationships — or changes in the body of the agent — increase in friction between actuators, malfunctions due to injuries, *etc*. Moreover, the set of changes has a long tail — changes can be caused by a once in a life-time pandemic (Heaven 2020). To be robust to these changes, an offline learned system has to anticipate them and take them into account when learning. An online learner, on the other hand, can afford to not worry about them until they have occurred.

Second, the performance of an online learning system is not limited by the quality of the data-set used for learning. The learner can gather more data by interacting with the world and fill gaps in its knowledge. Online data collection is especially important if the data-distribution has a long tail; capturing this long tail in an offline data-set cannot only be prohibitively time consuming and expensive but also wasteful — the agent might never encounter some rare cases in the long tail of the data.

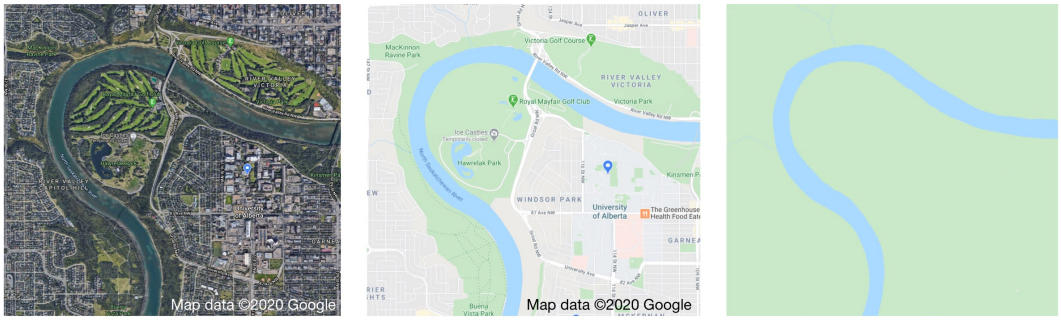
A real-world example of the challenges introduced by the long tail of the

data-distribution is the difficulties faced by Tesla AI for training a simple stop-sign detector. Even after collecting a large data-set of stop-signs using a fleet of cars, they observed that there were rare instances where the system failed. To fix this, they collected data of the failure cases, retrained the detector on the failure cases, and repeated the process many times to capture the long-tail of stop signs (Karpathy 2020). This human-in-the-loop learning achieved good results but was expensive and relied on human interventions. An online learner, on the other hand, can side-step these issues by adapting to rare cases online. For example, if there is a peculiar stop-sign on a route that a user takes — perhaps the sign is partially occluded by a tree and only detected by the learner when the car passes the tree — an online learner might miss the sign the first time it drives through the route — only seeing it after passing the tree. However, it can learn from its mistake and correctly detect the stop sign next time. An offline system, on the other hand, has to learn to simultaneously deal with the set of all peculiar stop-signs in the world, wasting useful representation capacity in the process.

In addition to learning online, an intelligent agent must learn abstractions from sensory data — representation learning — for achieving goals. (1) Computation and memory efficiency, and (2) generalization are the two main motivations for learning abstractions.

(1) A practical learner must use its limited capacity efficiently by discarding sensory information not useful for achieving goals. For example, if the goal of the agent is to learn to use banknotes for transactions, it is sufficient to learn to authenticate them and learn to discriminate between their denominations. All other details — such as pixel-level details and name of the founding-father on the note — can be discarded (Epstein 2016). This allows the learner to represent an abstracted view of the note using a fraction of the bits required to represent the actual note.

(2) Representation learning also enables generalization. Generalization is the ability of a learner to achieve goals in a part of the world it has never seen before. The new part of the world — despite being different from the parts the agent has visited in the past — might appear similar to something the



(a) All information (b) General purpose navigation (c) Kayaking

Figure 1.2: Depending on the goal of the agent, different abstractions of the same map are ideal for learning. A general-purpose road navigation agent needs to know about the location of roads and landmarks, but information about the position of the trees, and rooftops of buildings can be ignored. A kayaking agent needs even less information — a map of the waterways is sufficient. Operating on an abstracted view that only has the necessary information is both computationally and memory efficient. Moreover, by ignoring the extraneous information, the agent can be robust to changes that are not reflected in its abstracted view of the world, enabling generalization.

agent has seen in its abstracted view of the world. As long as the differences between the two parts are not important for achieving goals, the agent would generalize effectively. Examples of the two benefits of representation learning are depicted in Figure 1.2.

We have made significant progress for learning better neural network representations over the last decade. Neural networks trained with back-propagation have achieved impressive results on prediction tasks — such as image classification (Krizhevsky et al. 2012), natural language processing (Brown et al. 2020), object detection (Girshick 2015; Redmon et al. 2016; Ren et al. 2015) — and control tasks — learning to play Atari (Mnih et al. 2015), Starcraft (Vinyals et al. 2019), and DOTA (OpenAI 2018). Nonetheless, existing methods have two limitations: they are incompatible with online learning, and they do not learn representations explicitly optimized for online-learning.

Current representation learning methods require an accurate estimate of the gradient of a metric for updating the parameters of a neural network. Computing this gradient online is often not possible. For example, to get a low variance estimate of the gradient for a single learning update, OpenAI

Five and AlphaStar use 1 million (OpenAI 2018) and 65 thousand (Vinyals et al. 2019) observations respectively. Similarly, the largest GPT-3 language model uses 3.2 million samples for a single learning update (Brown et al. 2020).

Additionally, representations learned by existing algorithms are not optimized for online tracking. As I shall show in later Chapter 4, representations learned with the explicit goal of tracking — online-aware representations — are highly effective at online learning. Conversely, representations that are not online-aware can be poor for online adaptation even if they contain the same information as their online-aware counterparts.

Keeping in mind the necessity of online learning and representation learning, I am interested in answering two questions in this thesis: first, can neural networks learn online-aware representations — representations that are explicitly optimized for online learning, and second, can they learn these representations online.

The first question is agnostic to how the neural network is learned. I am simply interested in investigating if, in the function class of neural networks, there exists a solution — a set of network parameters — that can transform the input sensory data to a vector representation that enables effective online learning. The second question, on the other hand, pertains to how the neural network is learned — we not only want to learn online-aware representations but also learn these representations online.

The rest of the thesis is organized as follows. In Chapter 2, I will summarize the existing work on catastrophic forgetting, representation learning, and credit-assignment through time in neural networks. I will then define the online prediction learning problem and give an overview of my online learning architecture in Chapter 3. Chapter 4, will cover Online-aware Meta-Learning (OML) (Javed and White 2019) — a gradient-based method for learning online-aware representations that achieves impressive empirical results. Despite the success of OML, in Chapter 5, I will show that OML does not address the interference problem in neural networks. To address the limitations of OML, I will introduce Learning with Backtracking (LwB) in Chapter 6. LwB is a preliminary version of a general scalable online representation

learning method that does not suffer from interference. Finally, in Chapter 7, I will conclude by summarizing the key takeaways of the thesis and speculate a future direction for online representation learning.

Chapter 2

Preliminaries

In this chapter, I will give a brief overview of the broader context surrounding the work in this thesis. The broader context includes (1) a history of neural networks and the back-propagation algorithm, (2) catastrophic interference: reason neural networks have been ineffective for online learning, (3) a summary of current paradigms for representation learning methods for neural networks, and (4) an overview of the credit-assignment through-time problem.

2.1 A Brief History of Neural Networks

Neural networks are a function class loosely inspired by how the brain processes information. A neural network consists of a set of nodes — neurons — that are connected to neighboring nodes through incoming and outgoing edges. The edges are represented by tunable weights. Information enters a node from the incoming edges, is processed at the node, and forwarded to the outgoing edges based on some criteria. These nodes can be arranged in layers, and the layers can be cascaded to construct a deep neural network. Learning is driven by changes in the tunable weights on the edges. Information processing in neural networks is distributed across the nodes that only communicate through the edges. This distributed and cascaded structure of the neural networks makes them well suited for learning hierarchical representations.

Early work in neural networks was inspired by attempts at understanding the learning principles in biological brains. Hebb (1949) proposed Hebbian learning — the idea that changes in the strength of the connections between

neurons drive learning and if two nearby neurons fire at the same time, the connection between them is strengthened. Farley and Clark (1954) — motivated by Hebb's work — simulated non-linear networks in a computing machine, and showed some promising preliminary results. Later that decade, Rosenblatt (1958) proposed Perceptron, an algorithm for processing visual stimuli. Perceptron was inspired by how biological beings processed visual information and consisted of a single layer neural network — a linear classifier. It showed promising empirical results on visual tasks. Soon after, Minsky and Papert (1969) showed that despite the empirical success of the perceptron, it could not represent some rudimentary functions, such as an Exclusive-OR. They suggested that multi-layer neural networks might fix the limitation of the perceptron. The negative results from Minsky and Papert (1969) discouraged further research on neural networks, and for nearly a decade, no one knew how to extend perceptron to a multi-layer setting. It was not until back-propagation — independently discovered by Lecun (1985), Rumelhart et al. (1986), and Werbos (1974) — was proposed that the neural network community found a way to train deep neural networks.

Since then, the back-propagation algorithm, combined with more data (Deng et al. 2009), compute, and better models (Goodfellow et al. 2014; He et al. 2016; Hochreiter and Schmidhuber 1997; LeCun et al. 1998; Vaswani et al. 2017) has been the driving force behind the success of neural networks. Neural networks based systems have achieved impressive performance on a multitude of tasks (Brown et al. 2020; Moravk et al. 2017; OpenAI 2018; Silver et al. 2017; Vinyals et al. 2019) and continue to improve with increase in capacity and compute (Brown et al. 2020). Despite their success, however, they have remained ineffective for online learning. One of the main reasons is that neural networks trained with back-propagation suffer from *Catastrophic Interference*.

2.2 The Catastrophic Interference Problem

Catastrophic interference in neural networks is the tendency of back-propagation to over-write the existing knowledge stored in the parameters of a neural network. While it is expected that new learning will interfere with previous learning to some extent, the degree of interference in a neural network trained with back-propagation is crippling. Significant research — conducted both before and after the deep learning revolution — has tried to address the problem, but a robust and scalable solution is yet to be found.

The term *Catastrophic interference* was coined by McCloskey and Cohen (1989). They noticed that neural networks — when trained on a new task — performed poorly on previously learned tasks. They further showed that after sufficient training on a new task, the performance on older tasks was as bad as an untrained network, and relearning the older tasks a second time was not any faster.

French (1999) studied the phenomenon in more detail and argued that catastrophic interference in neural networks exists because of representation overlap and weight cliffs — small changes in the parametric space causing large changes in the output. If, while learning a new task, some of these cliffs are traversed, the output of the network on older tasks could change drastically, causing forgetting. French (1992) proposed node-sharpening as a solution to address forgetting. Node sharpening only marginally helped with interference by making neural network representations less distributed. More recently, Ghiassian et al. (2020) and Liu et al. (2019) proposed alternate ways for making neural network representations less distributed. Ghiassian et al. (2020) proposed mapping the input observation to a sparse high-dimension vector before passing it to a neural network whereas Liu et al. (2019) proposed regularizing the neural network activations to make them more sparse and less distributed. Both methods showed promising results for online learning and imply that making neural networks less distributed reduces interference.

Robins (1995) took a different approach and suggested storing older data for reducing interference. He showed that by interleaving data of older tasks

with the data of a new task, it is possible to significantly reduce forgetting. When older data is not available, he proposed using synthetic data representative of the older tasks instead.

Rehearsal based methods have been extensively studied since then. Generally, they store instances of the data-stream in a buffer or the parameters of a generative model; during learning, they interleave the most recent sample of data with older data sampled from a buffer (Aljundi et al. 2019; Chaudhry et al. 2019; Javed and Shafait 2018; Lin 1992; Mnih et al. 2015; Rebuffi et al. 2017; Riemer et al. 2019; Schaul et al. 2015) or generated using a model (Shah et al. 2018; Shin et al. 2017), and update the representation using a combination of old and new data. More recent work has explored the possibility of storing data selectively (Aljundi et al. 2019; Lopez-Paz and Ranzato 2017; Rebuffi et al. 2017); however, selective storage methods have shown to be only marginally helpful, if at all; the simplest implementations of memory buffers — based on recency for non-stationary problems and reservoir sampling for stationary problems — stay competitive (Chaudhry et al. 2019; Javed and Shafait 2018).

A different family of methods prevents interference by biasing the online update. They identify parameters of the model useful for older tasks, and regularize the weight updates such that weights important for older task are less plastic (Aljundi et al. 2018; Kirkpatrick et al. 2017; Lee et al. 2017; Zenke et al. 2017). These methods work well when it is possible to identify weights important for older tasks (Kirkpatrick et al. 2017). However, an effective method for identifying important parameters online is yet to be found.

Researchers have also looked at the human and animal brain for inspiration. McClelland et al. (1995) suggested that dual learning systems — a fast and a slower learner — are instrumental for mitigating interference. They argued that in humans and animals, learning first happens in a highly plastic hippocampal system and is then slowly transferred to the neocortex by interleaving experience of multiple tasks together. Inspired from their work, Ans and Rousset (1997) and French (1997) implemented dual systems using neural networks and showed that they were indeed effective for reducing inter-

ference to some extent. Decades later, Kemker and Kanan (2017) proposed a deep-learning version of a dual learning system that used a generative model to consolidate knowledge from a fast learner to a slow learner. Despite their biological motivation, these methods have not resulted in a generally effective solution; consolidating knowledge from a fast learner to a slow learner faces the same challenges as learning a single neural network system online.

2.2.1 The success of memory based methods

Among the above-mentioned families of methods, memory-buffer based methods have enjoyed the most success. For example, DQN (Mnih et al. 2015) — by employing large recency based buffers — is capable of off-policy Q-learning (Watkins and Dayan 1992; Watkins 1989) from pixels in Atari games (Bellemare et al. 2013). For every learning update, DQN samples an IID batch of data from its recency buffer for learning, effectively mitigating interference.

While DQN has achieved impressive results for online learning, it has several scalability challenges. The amount of data DQN stores in its buffer is orders of magnitudes larger than the number of parameters in the model and the horizon of the environment. For example, the Atari learning suite has a maximum episode length of 18,000 frames (Machado et al. 2018) and the DQN model has roughly 80,000 parameters (Mnih et al. 2015). The buffer, on the other hand, stores the equivalent of 7 billion parameters (1,000,000 samples of dimension 84 x 84). Even if the storage requirement of DQN scaled linearly with the horizon of the problem, the amount of storage needed for the buffer becomes prohibitively large very quickly. Reducing the size of the buffer adversely impacts the performance (Fedus et al. 2020).

An alternative to DQN for control is the on-policy actor-critic family of methods (Sutton et al. 2000). Actor-critic methods, when combined with back-propagation to learn deep networks, also suffer from catastrophic interference. One way of avoiding interference is to collect a large batch of on-policy data — by letting the agent run for a long time — before an update or by deploying multiple actors in parallel on copies of the environment (Mnih et al. 2016). These solutions are undesirable because an online learner only has access to a

single environment, and collecting a large batch of data for a single learning update prevents the learner from incorporating feedback from the environment in real-time.

To summarize, catastrophic interference is a bane when learning neural network representations online using back-propagation, and decades of research has not produced a promising solution.

2.3 Representation Learning in Neural Networks

Setting aside the catastrophic interference problem, significant progress has been made in designing representation learning methods for offline learning.

2.3.1 Supervised representation learning

The first big success of representation learning with neural networks was AlexNet (Krizhevsky et al. 2012). Krizhevsky et al. (2012) showed that a neural network trained by minimizing a supervised objective end-to-end on a large-scale data-set — Imagenet (Deng et al. 2009) — can learn powerful representations of the data. Yosinski et al. (2014) studied the transferability of Imagenet representations and found that when a new task is similar to the Imagenet classification task, models pre-trained on Imagenet perform much better than similar models trained on the new task from scratch. However, they also found that if the new task is significantly different, pre-training on Imagenet can hurt performance.

It is not surprising that representations learned by optimizing a supervisory signal are ineffective for transferring to radically different tasks. A neural network trained to minimize a supervisory signal has no incentive to keep information in the representation that is not useful for that supervised task. If a new task relies on information that is abstracted away in the representation, the representation would be a poor candidate for solving this new task. Yosinski et al. (2014) showed that one way around this issue is to fine-tune the last k layers of the deep neural network on the new-tasks — where k is

a hyper-parameter. They reasoned that the initial layers of a deep neural network extracted general-purpose representations that could be transferred to radically different tasks whereas the latter layers of the network were more task specific and needed to be fine-tuned on new tasks.

The success of deep supervisory learning provided the much-needed evidence to demonstrate the representational power of neural networks. However, supervised representation learning is not scalable. Labeling a large amount of data is expensive and time-consuming. Moreover, offline supervised learning methods assume that the input and target distribution is stationary, and cannot learn representations effective for non-stationary environments.

2.3.2 Unsupervised representation learning

To deal with the scalability issues of supervised learning, the deep learning community soon transitioned to unsupervised learning. Unsupervised learning methods, instead of relying on targets provided by experts, construct their target. These targets can be created by learning a generative model of the data-distribution (Goodfellow et al. 2014; Kingma and Welling 2013) or by target functions hand-crafted by human experts, such as in self-supervised learning.

One of the earliest attempts at unsupervised representation learning was to learn a compressed representation of the data using an auto-encoder (Hinton and Salakhutdinov 2006; Kramer 1991; Vincent et al. 2008; Vincent et al. 2010). An auto-encoder transforms the input data into a low dimensional embedding $\mathbb{R}^d$, and reconstructs the input from this low dimensional embedding. Once the learning has finished, the embedding can be treated as a representation capturing the important factors of the data. While auto-encoders are appealing — they are easy to understand and scale — reconstructing the observation in the input space — such as pixels — can force the representation to focus on unimportant pixel-level details. For example, Anand et al. (2019) showed that representations learned by VAEs (Kingma and Welling 2013) — a generative model built on the principles of auto-encoders — are no better than randomly initialized networks for one-step prediction tasks on the Arcade

learning environment (ALE) (Bellemare et al. 2013).

Self-supervised learning is a recently popularized subset of unsupervised representation learning. Instead of predicting targets given by human experts — as done in supervised learning — self-supervised learning proposes to predict auxiliary, task-independent, targets that are a function of the input observation. Given a data-point x, a self-supervised learning task consists of two functions f and g. The input to the model and the target label is given by $f(x)$ and $g(x)$ respectively, where both f and g are hand-designed to encode useful inductive biases. Some examples of self-supervised tasks are: extracting two patches from input observation, and predicting the relative location of the patches in the observation (Doersch et al. 2015); predicting the color of the pixels of an image (Zhang et al. 2016); rotating the input observation and predicting the degree of rotation (Gidaris et al. 2018); perturbing patches of images and forcing the network to classify them as belonging to the same class (Dosovitskiy et al. 2014); maximizing mutual information between input and the representation (Bachman et al. 2019; Hjelm et al. 2019) *etc*.

Self-supervised methods can achieve similar results as supervised learning for learning image classifiers, while using only a fraction of the labels (Chen et al. 2020; Hénaff et al. 2019; Oord et al. 2018). However, they rely on target functions that rely on expert knowledge and intuition. Moreover, self-supervised methods learn representations by making one-step predictions in the observation space; it is unclear if one-step predictions are sufficient for learning good representations, especially when verifying that a representation is good is a multi-step process.

Irrespective of the success and promise of unsupervised representation learning, unsupervised learning is not consistent with the idea that the goal of representation learning is to build abstractions for achieving goals more efficiently. Without any knowledge of the goal, a representation learning algorithm cannot distinguish between necessary and unnecessary information. Going back to our kayaking example in Figure 1.2, an unsupervised learning method is unlikely to abstract away details of the roads and buildings without prior knowledge of the goal.

2.3.3 Learning disentangled representations

A part of the representation learning community has looked at the notion of learning disentangled representations. The idea behind disentangled representations is to learn a set of variables that can capture independent factors of change in the data. For instance, the color and type of a car are independent of each other — it is possible to change one while keeping the other constant — and would be separated in a disentangled representation of a car. Proponents of learning disentangled representations argue that separating independent factors of change can lead to systematic generalization. Several unsupervised methods for learning disentangled representations have been proposed (Burgess et al. 2018; Higgins et al. 2017; Mathieu et al. 2016). However, Locatello et al. (2019) showed that coming up with a general unsupervised algorithm for disentangling factors of change is impossible. The only way to achieve disentanglement using an unsupervised method is to incorporate assumptions about the data distribution in the representation learning algorithm. Moreover, they showed — in a large-scale study — that existing unsupervised methods for disentangling representations do not succeed at learning these representations.

More recent work has looked at a more principled approach to learning disentangled representations. Bengio et al. (2020) proposed a meta-learning objective that exploits sparse interventions — changes in one of the independent factors of change — for disentangling mechanisms of the world. The central idea is that a disentangled representation can allow a learner to adapt to interventions quickly; conversely, optimizing for speed of adaptation can help a learner find these representations. Additionally, Bengio et al. (2020) proposed learning a causal structure between the learned abstractions of the world.

2.3.4 Representation search

Representation search is a less popular approach to representation learning. Instead of using back-propagation, the idea is to search for a representation

by continually generating new features — often randomly — and maintaining those that improve performance. Mahmood (2017) and Mahmood and Sutton (2013) showed that continually generating random features can improve representations on a simple domain. They further showed that feature search can be combined with back-propagation to improve over back-propagation alone.

These methods are promising because search is easy to scale, and can achieve impressive results on complex tasks (Silver et al. 2017). However, random-search could be extremely slow at discovering meaningful abstractions. To the best of my knowledge, representation search is yet to be demonstrated as a mechanism for learning representations in deep neural networks in a convincing manner.

2.4 Credit Assignment Through Time

The last bit of preliminary knowledge required for this thesis is the problem of credit-assignment through time. It can be informally defined as linking the decisions and components of a learning system with outcomes, especially when the outcomes are delayed in time (Minsky 1961). This problem arises in a multitude of situations: studying for a test can result in better grades, but the grades might not be announced for months after the test; storing food at room temperature might spoil the food but only after a few days; eating excessive junk food might increase risk of heart diseases, but the effect might not be observable for decades. In all these examples, linking the effect with the cause is not trivial, and requires pinpointing the actions and internal structure of the learner — among hundreds and thousands of actions and components — that led to the outcome.

Early work in credit assignment can be divided into two different lines of research. One looked at assigning credit of outcomes to actions while ignoring the internal decisions that led to the action and the other aimed to assign the credit of error in a prediction to the internal decisions of the learner. Sutton (1984) termed the former the *temporal credit assignment problem* and the latter the *structural credit-assignment problem*. If the target is a function

of a sequence of observations — such as language translation — the structural credit-assignment can involve assigning credit to an internal decision that happened multiple time-steps ago. Consequently, both the temporal and structural credit-assignment problem require credit-assignment through time.

For temporal credit-assignment, Temporal-difference learning (TD learning) is one of the most successful methods. The idea behind TD learning is to propagate credit from the outcome back to the action on every subsequent visit to the relevant states (Sutton 1988). A learner achieves this by bootstrapping its own belief about the world to create its targets. TD learning is light-weight, principled, and can do arbitrary long credit-assignment in a fully incremental way given enough interaction with the world. It can also be combined with other incremental methods, such as eligibility traces (Sutton 1984), that use heuristics based on recency and frequency of actions to assign credit. Finally, TD learning can be combined with planning (Sutton 1990) to do long-term temporal credit assignment without revisiting the states many times. One of the earliest successes of TD methods was TD-gammon (Tesauro 1995), a reinforcement learning system that learned to play backgammon. Backgammon, like most strategy games, requires temporal credit-assignments as moves can have delayed consequences, and TD learning provided an effective solution.

The solutions to structural credit-assignment through time have also enjoyed success over the past decades. The driving force behind this line of work is Back-propagation through time (BPTT) (Mozer 1989; Robinson and Fallside 1987; Werbos 1988; Williams and Zipser 1995). BPTT, combined with advances in recurrent neural network architectures (Cho et al. 2014; Hochreiter and Schmidhuber 1997), can be used to train recurrent neural networks that predict a target that is a function of multiple observations. For example, translation of a sentence from English to French depends on the complete sentence in English and not individual words. A learner that processes text at word level would need a mechanism to assign the credit of an error in translation to internal decisions made over multiple time-steps. BPTT has been used to train impressive systems for language translation (Bahdanau et al. 2014), and control (Kapturowski et al. 2018), among other tasks.

2.4.1 Challenges in scaling BPTT

Unlike TD, BPTT does not scale well with the delay between the internal decisions and the outcome; The most computationally efficient implementation of BPTT stores all the interval activation of a network at every time step. This means that for a network with m activations (not the same as weights), BPTT would require memory to store $m \times k$ activations, where k is the truncation window. This might seem doable — memory is cheap — but if our end-goal is to scale models to have trillions of parameters, requiring an order of magnitude more memory for a single learning update is not desirable.

The memory-requirement of BPTT can be reduced at the cost of an increase in computation time. For instance, Chen et al. (2016) proposed selectively storing activations of past time steps. The remaining activations can be recomputed when needed for propagating the gradients. Along similar lines, Gruslys et al. (2016) proposed a more general algorithm that can trade-off memory and computation to fit the needs of the user. However, even the most memory efficient implementation of BPTT — one for which the computation time grows quadratically with the horizon of propagation — requires storing at-least the sensory data of a sequence. Storing sensory data for arbitrary long sequences is also not scalable, especially for high-dimensional data. For instance, a 1080p HD video frame has dimensions $1920 \times 1080 \times 3$. Storing only a thousand of these frames in an uncompressed form would require roughly 6 billion bytes of storage.

In addition to memory issues, BPTT also suffers from vanishing and exploding gradients. Back-propagation requires multiplying the gradients multiple times with numbers that are often smaller than one. This causes the gradient to decay exponentially (Bengio et al. 1994; Pascanu et al. 2013), making learning prohibitively slow.

Some recent work has looked at attention mechanisms (Vaswani et al. 2017) for overcoming the vanishing gradient problem for arbitrary long sequences. Instead of back-propagating through all past sequences, (Ke et al. 2018) proposed an attention mechanism that can introduce skip connections in the neu-

ral network computation graph, essentially allowing the gradients to jump back multiple time-steps. They achieve skip-connections by using an attention mechanism to make the output a function of a sparse set of past inputs. Their method effectively addresses the vanishing gradient problem but does not address the memory concerns of BPTT. Follow-up work by (Kerg et al. 2020) addresses the memory concerns of BPTT to an extent. They propose using a screening mechanism to store *promising* input observations for assigning credit in the future. Their method assumes that credit for an outcome can be assigned to a sparse set of observations, and these observations can be identified in advance. This is true in many cases, but untrue for others. For instance, allocating limited capacity of a neural network to a large set of inputs is a credit-assignment problem that requires taking into consideration the complete input distribution. The capacity allocation problem can arise even if the inputs are Markovian, and the relevancy screening does not provide an adequate solution. Nonetheless, their work is a promising direction for scaling BPTT to arbitrary long horizons.

Chapter 3

Problem Formulation and Notation

I define the online prediction problem in terms of a Markov Process (MP). An MP is defined by $(\mathcal{S}, p)$, where $\mathcal{S}$ is the set of states, and $p : \mathcal{S} \to \mathcal{S} = P(S_{t+1} = s_{t+1} | S_t = s_t)$ is the transition model of the world. At every time step, the MP transitions from S_t to $S_{t+1} \in \mathcal{S}$ following p.

3.1 Problem Formulation

The online prediction problem consists of a Markov process $(\mathcal{S}, p)$ and a non-stationary target function $f_t : \mathcal{S} \to \mathcal{Y}$. The goal of the learner is to predict the target $y_t \in \mathcal{Y}$ given by $f_t(S_t)$, where S_t is the state at time-step t. At time t, the learner outputs $\hat{y}_t$, an estimate of y_t, and incurs a loss given by $\mathcal{L}(y_t, \hat{y}_t)$, where $\mathcal{L}$ is a cost function that returns the prediction error of the prediction. As a result, the MP transitions to a new state S_{t+1} and the learner receives the true target $f_t(S_t) = y_t$ from the world.

Given this setting, we define the goal of the learner as minimizing the error accrued over time. Let $\hat{f}_{\theta_t}$ be the learner's estimate of f_t, where θ is a set of tunable parameters spanning a class of functions. The accumulated error up to time T is defined as:

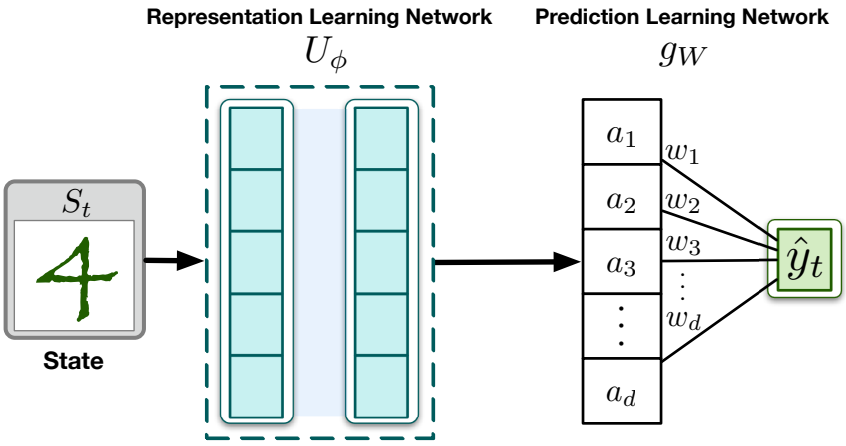


Figure 3.1: An architecture for online learning. I decompose the online prediction function into two components — a Representation Learning Network (RLN) and a Prediction Learning Network (PLN). The RLN aims to learn a representation $\mathbb{R}^d$ of the state that is effective for online tracking. Given $\mathbb{R}^d$, the PLN tracks the target y_t. PLN is updated at every step to incorporate feedback from the environment in real-time, and should represent a method that can effectively learn online — such as a linear regression predictor.

$$Loss_T = \sum_{t=1}^{T} \mathcal{L}(f_t(S_t), \hat{f}_{\theta_t}(S_t)) \tag{3.1}$$

$$= \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y}_t) \tag{3.2}$$

An alternative formulation to Equation 3.1 would be to minimize dynamic regret (Zinkevich 2003) by subtracting the best comparator from the loss at each step. Regret removes the irreducible error of the problem and provides a more interpretable metric; however, computing dynamic regret requires computing a sequence of optimal comparators which is not trivial for a function class spanning deep neural networks. Nonetheless, because minimizing accumulated loss also minimizes dynamic regret, I will use the term regret minimization to refer to the goal of the learner even though the learner will never compute the actual regret. I provide the dynamic regret based formulation of the problem in Appendix A.1.

The MP formulation of the problem might seem limiting but is sufficient to study catastrophic interference in neural networks. The solution methods

introduced in this thesis are general and can be extended to an MDP or a Partially Observable MDP (POMDP). With the POMDP generalization, an online prediction problem can represent important research problems, such as learning a model of the world for planning (Sutton 1990), or learning General Value Functions (GVFs) (Sutton et al. 2011). Additionally, by making the target function f a function of the parameters of the learner — a function that returns bootstrapped estimate of the return of a value function — the online prediction problem can represent minimizing the online temporal difference error (Sutton 1988). I describe the POMDP generalization of the problem in Appendix A.2.

Finally, it is pertinent to discuss the kind of non-stationarities f can have for effective learning to be possible. While I do not formally model the non-stationarities f can have, it is clear that tracking an arbitrary non-stationary target is not possible. f either has to change slowly or locally for learning to be effective. The benchmark I consider in this thesis will use a target function that both changes slowly and locally *i.e.*, only target associated with a few states changes every few steps.

3.2 An Architecture for Online Learning

The regret minimization problem defined above is fairly broad and admits a wide variety of solution methods. In this thesis, I am going to explore a particular family of methods that learn *Online-aware Representations*. Defining online-aware representations requires additional notation as follows: I decompose $\hat{f}$ into two functions, g and U, parameterized by W and ϕ respectively such that:

$$\hat{f}_\theta(S) = g_W(U_\phi(S)). \tag{3.3}$$

$U_\phi : \mathcal{S} \to \mathbb{R}^d$ — called the Representation Learning Network (RLN) — outputs a d-dimensional representation of the state. $g_W : \mathbb{R}^d \to \mathcal{Y}$ — called the Prediction Learning Network (PLN) — takes the representation given by the RLN and outputs $\hat{y}$, the estimate of the target. The decomposition of θ is

shown in Figure 3.1. RLN and PLN can be learned using different learning algorithms. Let Q^{PLN} be the learning rule used to update the PLN, g_W, *i.e.*,

$$W_{t+1} = Q^{PLN}(\phi, S_t, W_t, \mathcal{L}(y_t, \hat{y}_t), \nabla_{W_t}\mathcal{L}(y_t, \hat{y}_t)), \tag{3.4}$$

then my architecture for online learning constraints the space of solution methods by requiring the learner to update the PLN at every time-step using Equation 3.4. This constraint divides the solution method into two parts: (1) an incrementally learning PLN that can adapt to changes quickly by incorporating feedback from the world in real-time and (2) an RLN that is not constrained to update in real-time and can learn slowly over-time. The constraint also rules out methods that wait and collect a batch of data for updating the PLN. Finally, the learning method for the RLN has to deal with a continually changing PLN. I call an RLN learning algorithm that takes into account a changing PLN *Online-aware* and the resulting state representation an *Online-aware Representation* of the state. Representation learning methods described in Chapter 2 — with the exception of representation search and the disentangled representation learning work of Bengio et al. (2020) — are not online-aware.

Chapter 4

Learning Online-aware Representations using Gradients

In the previous chapter, I discussed the online prediction problem and an architecture for learning online-aware representations. In this chapter, I will introduce a gradient-based algorithm for learning online-aware representations for deep neural networks. The algorithm — called Online-aware Meta-learning (OML) — learns a representation by exploiting the fact that if the loss is twice differentiable, the update function used by the PLN — Q^{PLN} — is itself often differentiable with respect to the parameters of the network. For such update functions, we can compute the gradient of the parameters of the RLN — ϕ — to minimize the regret.

4.1 Related Work

OML uses meta-gradients (Bengio et al. 1991; Sutton 1992) for learning online-aware representations. Recently, meta-learning approaches with gradient descent have been introduced particularly for representation learning (Finn et al. 2017a; Li et al. 2017), which is the most relevant to OML. In this section, I describe this related work. Finn et al. (2017a) proposed MAML — an algorithm for fast adaptation to changes in the data distribution by exploiting the fact that an SGD update is differentiable. MAML assumes that the learner is interested in solving a set of mutually exclusive tasks given by a distribution $\mathcal{D}$ over tasks. Given this distribution, it aims to learn a network initialization

such that starting from this initialization, the network can adapt to any task $T_i \sim \mathcal{D}$ using a few steps of SGD. A single leaning step of MAML consists of (1) sampling a task $T_i \sim \mathcal{D}$, (2) sampling two sets of observations and targets — $(X_{train}, Y_{train}), (X_{val}, Y_{val})$ — from T_i, (3) updating the network to minimize error on (X_{train}, Y_{train}) using a few SGD steps, (4) measuring the validation error of the updated network on (X_{val}, Y_{val}), and finally, (5) updating the network initialization — the parameters of the network before step 3 — to minimize the validation error by differentiating through the few steps of SGD. After sufficient learning, MAML can successfully adapt to a task in $\mathcal{D}$ using few steps of SGD. One intuition behind the success of MAML is that the learned initialization lies close to the solution manifolds of all the tasks, and moving quickly to any of these manifolds is possible using only a few steps of gradient updates computed using a small number of samples.

Similar to MAML, OML also differentiates through the learning update but differs in some important ways. First, OML does not aim to learn a network initialization. Instead, the goal is to learn the RLN such that PLN achieves low regret. This enables OML to incorporate principled incremental learning algorithms in PLN. Secondly, OML does not assume a distribution over tasks and operates directly on an online stream of data.

Bengio et al. (2020) simultaneously proposed an approach similar to OML for learning representations using gradient-based meta-learning. They proposed a meta-objective that maximized the speed of adaptation to distributional shifts caused by sparse interventions on the data-generating factors. Using this objective, they proposed to learn an encoder — similar to the RLN — that transformed the input into a representation that was conducive to fast adaptation. In addition to an encoder, they also proposed to learn a causal structure on the output of the encoder. Their solution method and OML share similarities, but are solving different problems: their goal is to discover causal variables and the relations between these causal variables from sensory data, whereas OML aims to minimize regret on an online prediction problem.

OML also shares similarities with IDBD (Sutton 1992) and its follow-up work (Schraudolph 1999; Veeriah et al. 2017). IDBD (Sutton 1992) uses meta-

gradients to update the per-parameter step sizes for a linear predictor. By changing the step-size of a feature, IDBD can control how important a feature is for learning. Schraudolph (1999) extended IDBD by learning step-sizes for non-linear networks. Veeriah et al. (2017) proposed Crossprop that uses meta-gradients for learning the incoming weights of a single hidden layer neural network. Unlike OML, however, Crossprop is limited to networks with a single hidden layer. Moreover, Crossprop does not take into account multiple Q^{PLN} updates for computing the meta-gradient.

Algorithm 1: Online-aware Meta-learning

Require: $\mathcal{L}$: Loss function.
Require: S_1: Initial state.
Require: T: Total interactions with the MP.
Require: k: Parameters controlling number of BPTT steps.
Require: α: Learning rate for the representation update.
Require: U_{ϕ_1}, g_{W_1}: RLN and PLN parameters initialized as desired.
Require: Q^{PLN}: Learning rule for updating the PLN.

$i = 1$
$m = 1$
while $i < T$ **do**
  $L_{inner} = 0$
  for $j = 1, 2, \cdots, k$ **do**
    $\hat{y}_i = g_{W_i}(U_{\phi_m}(S_i))$
    $y_i, S_{i+1} = \{\text{Environment returns the target and the next state}\}$
    $y_i = f_i(S_i)$
    $L_{inner} = L_{inner} + \mathcal{L}(y_i, \hat{y}_i)$
    $W_{i+1} = Q^{PLN}(\phi_m, S_i, W_i, \mathcal{L}(y_i, \hat{y}_i), \nabla_{W_i}\mathcal{L}(y_i, \hat{y}_i))$
    $i = i + 1$
  end for
  $\phi_{m+1} = \phi_m - \alpha \nabla_{\phi_m} L_{inner}$
  $m = m + 1$
end while

4.2 Online-aware Meta-learning

The central idea behind OML is to update the RLN — U_ϕ — so that the learned representation minimizes the regret achieved by the PLN — g_W. Let Q^{PLN} be differentiable with respect to the parameters of the RLN. Moreover, let $Loss_{i:j} = Loss_j - Loss_{i-1}$ be the loss incurred by the learner from time-step i to j, where $Loss_i = \sum_{t=1}^{i} \mathcal{L}(f_t(S_t), \hat{f}_{\theta_t}(S_t))$.

At time-step c the learner's PLN and RLN will have parameters W_c and ϕ_m, respectively. The learner can apply Q^{PLN} k times to get $W_{c+1}, W_{c+2}, \cdots W_{c+k}$. Finally, it can then update the RLN from ϕ_m to ϕ_{m+1} by minimizing $Loss_{c:c+k}$ as follows:

$$\phi_{m+1} = \phi_m - \alpha \nabla_{\phi_m} Loss_{c:c+k} \tag{4.1}$$

$$= \phi_m - \alpha \nabla_{\phi_m} \sum_{t=c}^{c+k} \mathcal{L}(y_t, \hat{f}_{\theta_t}(S_t)) \tag{4.2}$$

$$= \phi_m - \alpha \nabla_{\phi_m} \sum_{t=c}^{c+k} \mathcal{L}(y_t, g_{W_t}(U_{\phi_m}(S_t))) \tag{4.3}$$

where α is the learning rate for the RLN update. Recall that

$$W_{t+1} = Q^{PLN}(\phi_m, S_t, W_t, \mathcal{L}(y_t, \hat{y}_t), \nabla_{W_t} a\mathcal{L}(y_t, \hat{y}_t)) \tag{4.4}$$

and the value of W at each time step is tied to ϕ; as a result, computing the gradient in Equation 4.1 requires propagating gradients backward in time, similar to BPTT. The computation graph showing the flow of gradients for an OML update is shown in Figure 4.1 and the pseudo-code for the OML is given in Algorithm 1. OML uses Equation 4.1 to update the RLN once every k steps, where k is a tunable hyper-parameter.

4.3 Experimental Evaluation

I evaluate the performance of OML on an online regression benchmark. The aim of the following experiments is to provide clear results and not to demonstrate the performance of OML on complex tasks. Results on larger benchmarks — omniglot and mini-imagenet — using deep-convolutional networks are in the original OML paper (Javed and White 2019).

4.3.1 Baseline methods

I compare OML to the following baselines.

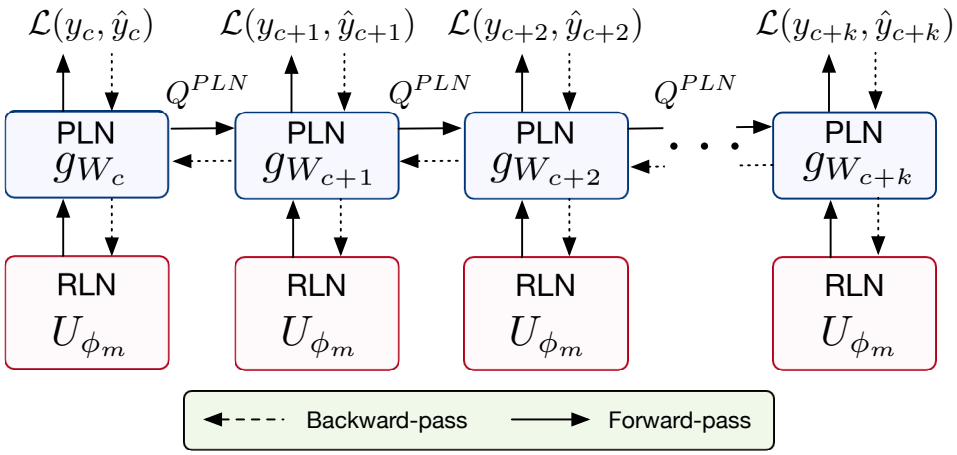


Figure 4.1: Computation graph for computing gradients for a single update of RLN. g_W is updated for k time-steps using the Q^{PLN} function, accumulating regret (or loss) for these k steps. If Q^{PLN} is differentiable, we can compute gradients for ϕ to minimize this regret using back-propagation through time (BPTT).

Online-SGD

Online-SGD uses the gradient at every time-step to update both the RLN and PLN using the Adam optimizer (Kingma and Ba 2015). Online-SGD is expected to suffer from catastrophic forgetting.

ER-RLN

ER-RLN uses experience replay to update the RLN, similar to DQN (Mnih et al. 2015); unlike DQN, however, the targets for the data in the buffer cannot be updated using a target network. This poses a challenge as outdated targets would interfere with newer targets when learning. To avoid this issue, I only update the RLN using the data in the buffer whereas PLN is learned using Q^{PLN} — similar to OML — using the most recent data. This approach is similar to the two-timescales architecture proposed by Chung et al. (2018) and allows the ER-RLN to not suffer from catastrophic forgetting problem for learning the RLN, while still tracking changes in the environment using the PLN. However, the RLN update of ER-RLN does not take into account the learning algorithm used by the PLN and is not *online-aware*.

Random-RLN

Random-RLN uses a randomly initialized RLN that is kept fixed throughout learning. It serves as a sanity check.

4.3.2 Online sine regression benchmark

The Online Sine Regression (OSR) benchmark consists of a Markov Process (MP) and a non-stationary target function. The state S_i of the MP is a vector of length 51. The first element of the state-vector is a real number in $[-5, 5]$ whereas the remaining fifty elements encode an integer between 1 and 50 in one-hot encoded form *i.e.*, one number in the last 50 elements of the state-vector is 1 and the rest are zero. S_{i+1} is generated by uniformly sampling the first element from $[-5, 5]$ and changing the number i encoded in the last 50 values of the state-vector to a randomly chosen $j \in \{0, 1, \cdots, 50\}$ with 0.2 probability. i stays unchanged with the remaining 0.8 probability.

The target function f is a combination of fifty sine functions — $\mathcal{T} = T_1, T_2, \cdots T_{50}$. If a state S encodes i in its last 50 elements, the target for the state is given by $T_i(S[0])$, where $S[0]$ is the first element of the state-vector. Function $T_i(x) = a_i \times sin(x + \phi_i)$ is generated by uniformly sampling its amplitude a_i from $[0.02, 6]$ and phase ϕ_i from $[0, \pi]$. The loss function $\mathcal{L}$ for the benchmark is the average mean-squared error between the target and the prediction. An example of the state-vector and the associated target is described in Figure 4.2.

The non-stationarity in the target function is created by replacing a randomly chosen T_i with $T_i^{'}$ at each step with 0.04 probability, where $T_i^{'}$ is generated by sampling its amplitude $a_i^{'}$ from $[0.02, 6]$ and phase $\phi_i^{'}$ from $[0, \pi]$.

This benchmark is similar to the few-shot regression benchmark introduced by MAML (Finn et al. 2017b) with some key differences. First, it is not episodic. Second, the changes in the target distribution are local and gradual. On average, one of the functions T_i changes every 25 steps. Finally, the input includes an additional variable — the function id — that is used to select the target. Achieving low regret on this benchmark requires learning to adapt

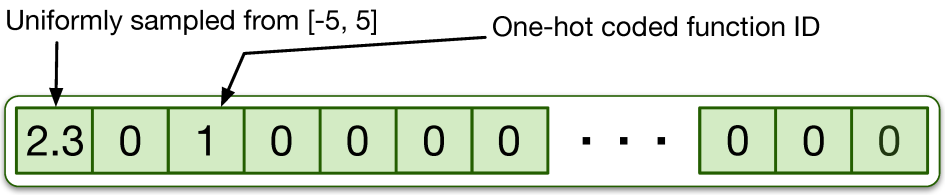


Figure 4.2: The state-vector for the OSR benchmark. The first element in the vector is a number uniformly sampled from [5, -5], whereas the remaining elements encode a one-hot encoded number between 1 and 50. In the picture, the encoded number is 2. That means the target for this observation is given by $T_2(2.3) = a_2 \times sin(2.3 + \phi_2)$, where 2.3 is in radians.

to local changes in the target distribution quickly while keeping the knowledge of the stationary part of the distribution intact. The few-shot learning benchmark introduced in MAML, on the other hand, only involves adapting to changes and does not require preserving old knowledge. The remaining details of the parameters of the benchmark, such as range of amplitude and phase, are the same as introduced by MAML (Finn et al. 2017a).

Implementation Details

The RLN is implemented using a deep neural network. It has 5 fully-connected layers, each with a width of 400, resulting in roughly 660,400 parameters. Each fully-connected layer is followed by a ReLU activation (Glorot et al. 2011). The output of the RLN is a vector $r \in \mathbb{R}^{400}$.

The PLN has 400 parameters that linearly combines the features in r to predict the target. All parameters are initialized using the method proposed by He et al. (2015) to prevent vanishing gradients due to the depth of the network.

The learning update for PLN — Q^{PLN} — is an online SGD update with adaptive per-parameter learning rates, and an l_1 penalty on the parameters of the PLN (Bengio 2017).*i.e.*

$$W_{t+1} = W_t - \boldsymbol{\gamma} \nabla_{W_t} (g_{W_t}(U_{\phi_m}(S_t)) - y_t)^2 + ||\boldsymbol{\omega} W_t||_1^1 \tag{4.5}$$

where $\boldsymbol{\gamma}$ is a vector with per-parameter learning rates and $\boldsymbol{\omega}$ is a vector that controls the l_1 regularization strength of each parameter. $\boldsymbol{\gamma}$ and $\boldsymbol{\omega}$ are initial-

Table 4.1: Hyper-parameters tried fro each method. RLN lr and $\boldsymbol{\gamma}$ are the learning rates used to update U_ϕ and g_W respectively . Online-SGD uses the same lr for both whereas the other two methods start with a $\boldsymbol{\gamma}$ of 3×10^{-3} and adapt it online. $\boldsymbol{\omega}$ controls the l_1 regularization strength on the parameters of the PLN, and is adapted online starting from a value of 1. All methods use the Adam optimizer (Kingma and Ba 2015) to update the RLN whereas Online-SGD uses Adam to updated both the RLN and the PLN. B_1 and B_2 are decay rates for the first-order and second-order running moment estimates of the gradients, and are used by Adam. Finally, ER-RLN has additional parameters to control the size of the buffer. These are (1) size of the buffer, (2) the ratio of buffer updates for learning RLN to the online updates to PLN, and (3) the size of the mini-batch of data sampled from the buffer for a learning update.

	Name of the Method		
Parameters	Online-SGD	ER-RLN	OML
RLN lr	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$	$1^{-2}, 1^{-3}, 1^{-4}, 1^{-5}$	$1^{-2}, 1^{-3}, 1^{-4}, 1^{-5}$
$\boldsymbol{\gamma}$	N/A	Adapted online	Adapted online
B_1	0.9, 0.95, 0.99	0.9, 0.95, 0.99	0.9, 0.95, 0.99
B_2	0.9, 0.95, 0.99	0.9, 0.95, 0.99	0.9, 0.95, 0.99
Buffer size	N/A	$10^3, 10^4, 10^5$	N/A
Mini-batch size	N/A	1, 16, 32 , 64	N/A
PLN/RLN ratio	N/A	0.1, 0.5, 1, 4 , 16	N/A
$\boldsymbol{\omega}$	$\mathbf{1^{-2}}, \mathbf{1^{-3}}, \mathbf{1^{-4}}, \mathbf{1^{-5}}$	Adapted online	Adapted online

ized to be a vector with each element set to 3×10^{-3} and a scaler 1, respectively. Both $\boldsymbol{\gamma}$ and $\boldsymbol{\omega}$ parameters are included in ϕ — the parameters of the RLN. Including $\boldsymbol{\gamma}$ and $\boldsymbol{\omega}$ in ϕ allows online adaption of step-sizes as well as regularization strength using meta-gradients. This online adaptation of step-sizes is similar to a multi-step version of IDBD (Sutton 1992). k in Equation 4.1 is equal to 25.

The hyper-parameters of all the methods are tuned independently using a grid-search using ranges described in Table 4.1.

Results

I report the results by first finding the best set of parameters using a grid-search over all parameters, and then running the best configuration with 30 random seeds. The averaged results are shown in Figure 4.3. OML achieves significantly lower regret compared to the baselines. OML also has the fewest hyper-parameters that need to be tuned. Buffer-based ER-RLN fails to learn meaningful representations, performing only marginally better than a Random-RLN. I suspect that the outdated data in the experience replay buffer can hurt performance more than help. Moreover, ER-RLN is not learning Online-aware representations that take into account a changing PLN. Online-SGD, despite the catastrophic forgetting problem associated with online back-propagation, out-performs ER-RLN.

Is OML performing optimally? It is hard to say. While it is not possible to achieve zero loss due to non-stationarities in the target function, it is also not clear what is the best achievable performance for this benchmark. In the following section, I provide some evidence that OML is performing well because it is robust to interference.

4.3.3 Measuring robustness to interference of the learned representations

To understand why OML performs better, I study the representations learned by the RLNs of OML, Random-RLN, and Online-SGD. I use the RLNs learned for 25 million steps in the previous section for each method. Additionally, I

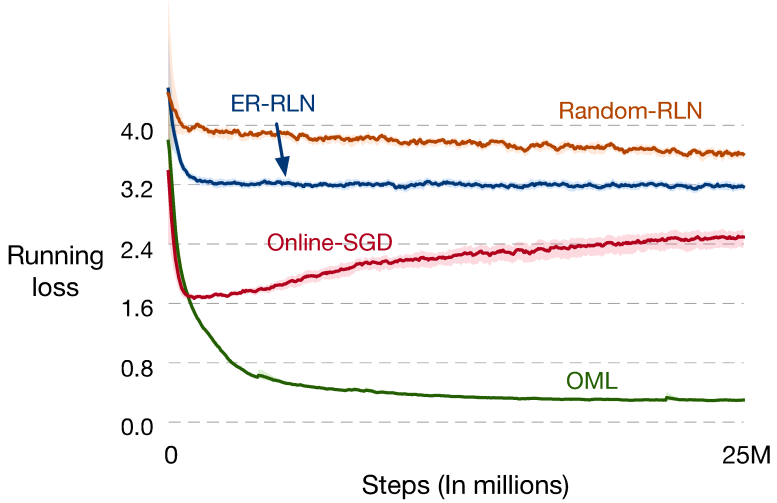


Figure 4.3: An exponentially decayed average of the loss for 25 million steps. The estimate of the loss is decayed by 0.97 at every step. OML achieves significantly lower running loss, and as a result, regret over 25 million steps. Experience-replay performs poorly because the target-distribution changes over-time and the outdated data in the replay buffer is not effective for learning whereas Online-SGD suffers from catastrophic forgetting. All results are averaged over 30 runs and the error bars are 95% confidence intervals created by 1,000 bootstrap samples.

create a new model — OML-Transformed — by linearly transforming the output of the RLN trained with OML. *i.e.* I transform the output of the RLN of the OML model, $r \in \mathbb{R}^{400}$, using a $M^{400\times400}$ — a 400×400 matrix. M is constructed to have ones in the diagonal, and real-numbers randomly sampled between $[0, 0.1]$ to fill the entries above the diagonal. All entries below the diagonal are zero. The resultant matrix is invertible and has a determinant of 1. Because the OML-Transformed representation is an invertible and linear transformation of the OML representation, both representations have the same information — a linear PLN trained till convergence on a prediction task on both representations would achieve similar results.

Experimental setup

I validate the robustness of representations learned by Random-RLN, OML, OML-Transformed, and Online-SGD to interference on a stationary prediction task of learning a set of fifty sine functions from a highly correlated stream

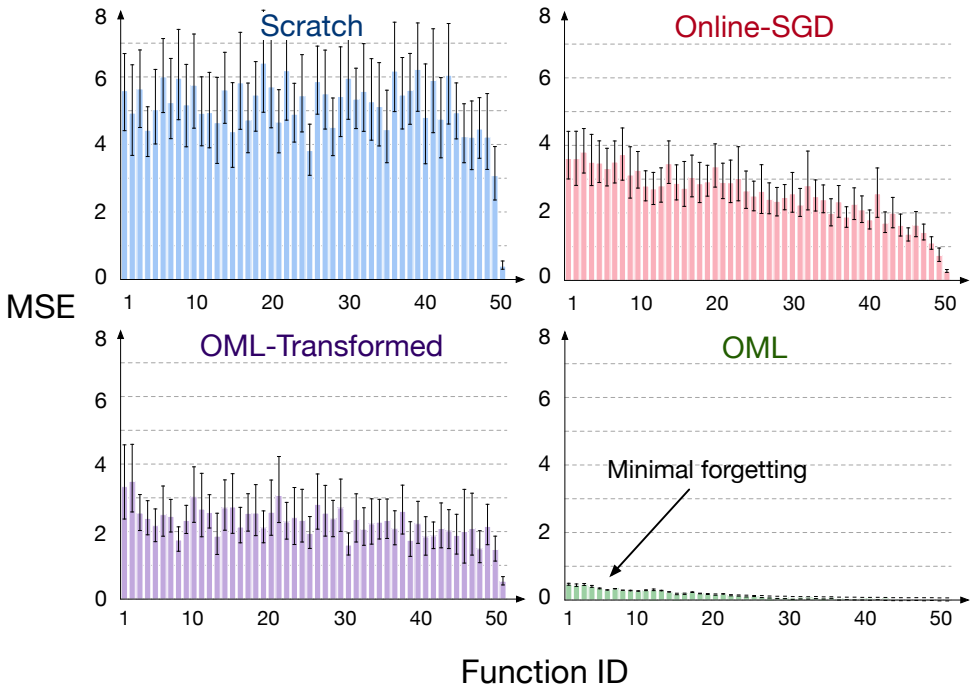


Figure 4.4: Error distribution on fifty functions after learning them from a correlated stream of data in a single pass. The function are seen in the same order as their ID. Error on earlier functions is higher due to interference from learning that happens afterward. Scratch representations are generated by a randomly initialized RLN. Apart from OML, all methods are incapable of learning without forgetting. OML-Transformed uses representations that are an invertible and linear transformation of OML representation but is still incapable of learning without forgetting. The poor performance of OML-Transformed is incompatible with the idea that representations can be evaluated by measuring how good they are at linearly disentangling factors of interests (Alain and Bengio 2017; Anand et al. 2019; Chen et al. 2020)

Table 4.2: Values of hyper-parameters tried for each method for investigating how robust representations learned by different methods are to interference. I report the result on the best configuration for each method.

Hyper-parameters	Values
Learning rate	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$
l_1 strength	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$
B_1	0.9, 0.95, 0.99, 0.995
B_2	0.9, 0.95, 0.99, 0.995

of data in a single pass. I construct the fifty sine functions, $T_1, \cdots T_{50}$, using the same procedure as before — sampling their amplitude from [0.02, 5] and phase from [0, π] — and create a single trajectory D_{traj} that consists of 30 input target pairs from T_1 followed by 30 pairs from T_2 and so on. The input distribution is the same as the OSR benchmark. For each of the methods, I initialize the PLN to be zero and learn it online on D_{traj} one sample at a time using gradient-descent. I evaluate the end performance of the PLN on an independent set of test data containing input target pairs from all fifty functions. I tune the learning rate and l_1 penalty of each method by doing a grid search over the values described in Table 4.2, and report the results with hyper-parameters that achieved the lowest test error averaged across all functions. This experiment represents an extreme case of learning a stationary distribution from a highly correlated stream of data in a single pass.

Results

I report the distribution of test error for all fifty functions in Figure 4.4 after learning on D_{traj}. Each experiment is repeated 50 times for a different set of 50 functions, and the error bars represent 95% confidence intervals constructed by 1,000 bootstrap samples. The error on earlier functions is higher because they suffer from more interference from future learning.

Random-RLN, Online-SGD, and OML-Transformed suffer from catastrophic interference and only achieve a low error on the last function — T_{50}. OML, on the other hand, can learn all fifty functions in a single pass with little forgetting. The discrepancy between OML and OML-Transformed is especially

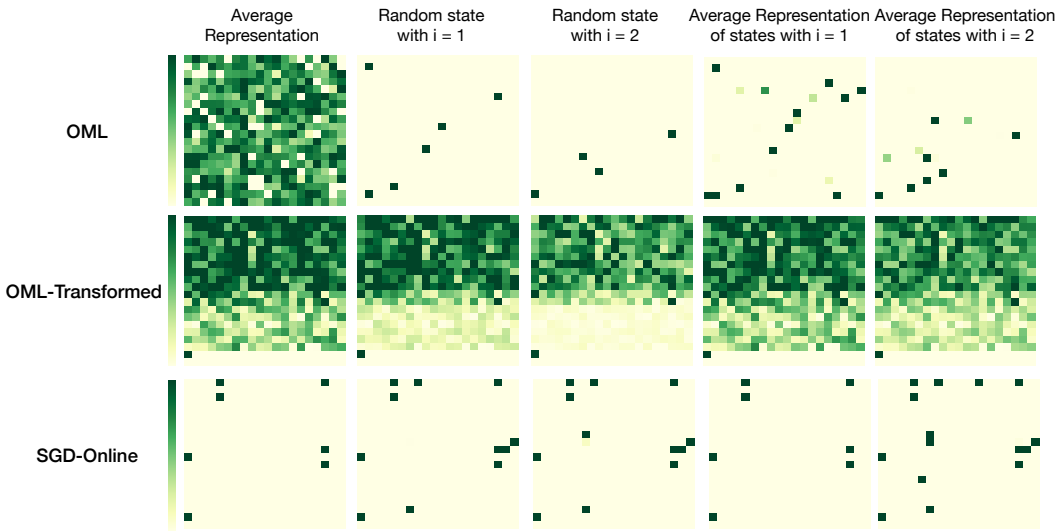


Figure 4.5: Visualization of representations learned by the RLN of OML, and SGD-Online. OML-Transformed representations are a linear transformation of the OML representations. OML learns highly sparse state representations. Additionally, representations for states that encode a different value of i do not overlap. This allows a learner to adapt to a change in part of the target function without impacting the knowledge associated with targets of other states. SGD-Online also learns sparse representations, but the representations for states with different targets overlap. In-fact — as shown in the first column — a large part of the representation space is not used by the SGD-Online to represent any state. OML, on the other hand, effectively uses the complete representation space while still achieving sparse state representations.

illuminating; it shows that representations that capture the same information — are rank-preserving linear transformations of each other — can result in very different performance. This goes against conventional representation learning wisdom that evaluates the quality of a representation by linear probing — training linear predictors till convergence on the representation (Alain and Bengio 2017; Anand et al. 2019; Chen et al. 2020). Linear probing would be unable to distinguish between OML and OML-Transformed representations.

4.3.4 Visualizing the learned representations

Finally, I visualize the representations learned by the RLN of OML, SGD-Online, and OML-Transformed. Recall that the state has an integer i encoded in the last 50 values that is used to select the target T_i. I visualize three kinds of representations: (1) the average representation of 5,000 random samples

from the complete input distribution, (2) the representation of two random samples, one with i equal to 1 and other with 2, and (3) and the average representation of 500 random samples with i equal 1 and 2. The representations are visualized in Figure 4.3.3. OML learns representations that are highly sparse for a single state. Moreover, the representations of states for which i is 1 have minimal overlap with states for which it is 2. The lack of overlap in the representation space explains why a PLN trained on the OML representations does not suffer from interference. Transforming this representation linearly — OML-Transformed — loses this sparsity property. Finally, while Online-SGD also learns sparse representations, the representations for states with different targets have significant overlap. Online-SGD also does not use a large part of the representation space to represent any state, wasting useful capacity.

4.4 Closing Discussion on OML

The key take-away from OML is that good representations for online learning are unlikely to emerge unless representation learning optimizes for online learning. This is evident from the fact that two representations with the same information — one is an invertible linear transformation of the other — can result in very different performance. Most of the existing representation learning methods, such as supervised learning, self-supervised learning, and generative modeling, learn representations by making one-step predictions that are insufficient for differentiating between the OML and OML-Transformed representations. It is unlikely that these methods can robustly find representations similar to OML.

The strong performance of OML also suggests that neural networks are capable of learning representations for continual learning. In addition to being effective on the simple OSR benchmark, OML is also effective at learning non-interfering representations from high-dimension data (Javed and White 2019), and has been extended to learn attention based architectures (Beaulieu et al. 2020). The OML objective has also been promoted as a solution to catastrophic interference by follow-up work (Johnson 2020). However, suggesting

that OML mitigates the catastrophic forgetting problem in neural networks is misleading. It merely shows that neural-networks can learn online-aware representations, and that learning online-aware representations is important. In the next chapter, I will show the OML updates suffer from interference when learning from a correlated stream of data, and use the results to motivate a family of methods that verify representation updates online before committing to them.

Chapter 5

Limitations of Gradient-based Representation Learning

In the previous chapter, I showed that OML — a gradient-based online-aware representation learning method — is effective at tracking non-stationarities using large deep neural networks. OML does this by learning a representation that mitigates catastrophic interference. However, Online Sine Regression (OSR) — the benchmark used to demonstrate the success of OML — had a limitation: it did not involve long-term credit assignment. In OSR, the transition function of the environment allows jumping to an arbitrary state from any state. Reducing interference between as few as two PLN updates is sufficient to reduce interference between arbitrary pairs of states. What happens when some of the states are temporally distant and lie either beyond the truncation window of the OML update or far enough that gradients vanish (Hochreiter et al. 2001)? I speculate that OML will not effectively reduce interference between such states and the OML update would suffer from catastrophic interference.

5.1 Catastrophic Interference in OML Updates

To test the robustness of OML to interference, I evaluate if it converges on a stationary problem.

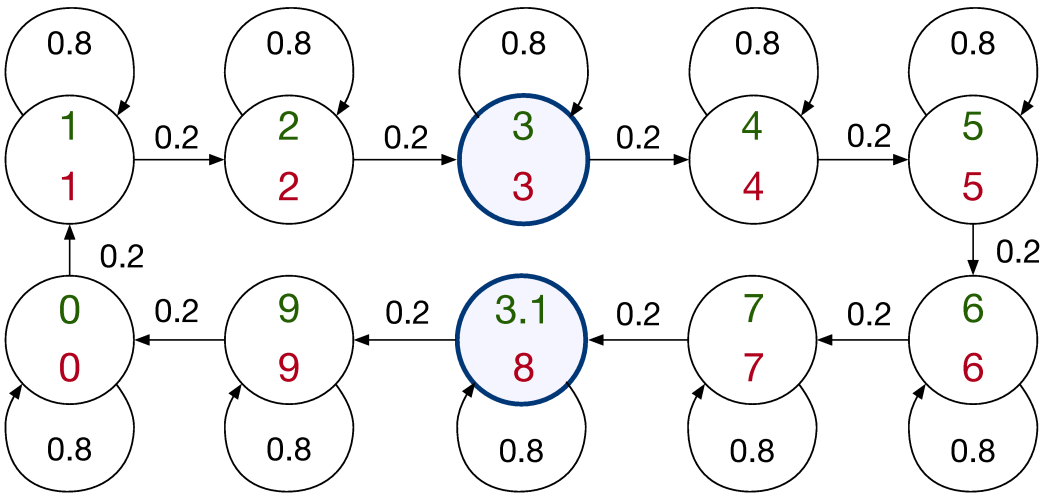


Figure 5.1: A ten-state Markov Process for evaluating the robustness of OML to interference. The top-green number in every circle is the state whereas the bottom-red number is the target. The arrows represent the transition function. The two shaded states — 3 and 3.1 — are close in the input space, but have different targets. A neural network trained online in this environment would have to deal with interference between the targets of states 3 and 3.1.

5.1.1 Ten-state Markov Process

I create a ten-state Markov Process (MP) with stationary targets. The MP is designed such that two temporally distant states are similar in the input space, but have different targets. A learner must learn a representation in which the features of these states do not overlap for mitigating interference. The environment is graphically shown in Figure 5.1. Each circle in the figure corresponds to a state-target pair. The top-green number is the state, whereas the bottom-red number is the target. The state 3 and 3.1 are close in Euclidean space but have different targets and can result in interference. The learner transitions to a new state every 5 steps on average and the average distance between the states 3 and 3.1 is 25 steps. Unlike the OSR benchmark, the learner cannot jump to an arbitrary state from any state and must learn from a correlated stream of data.

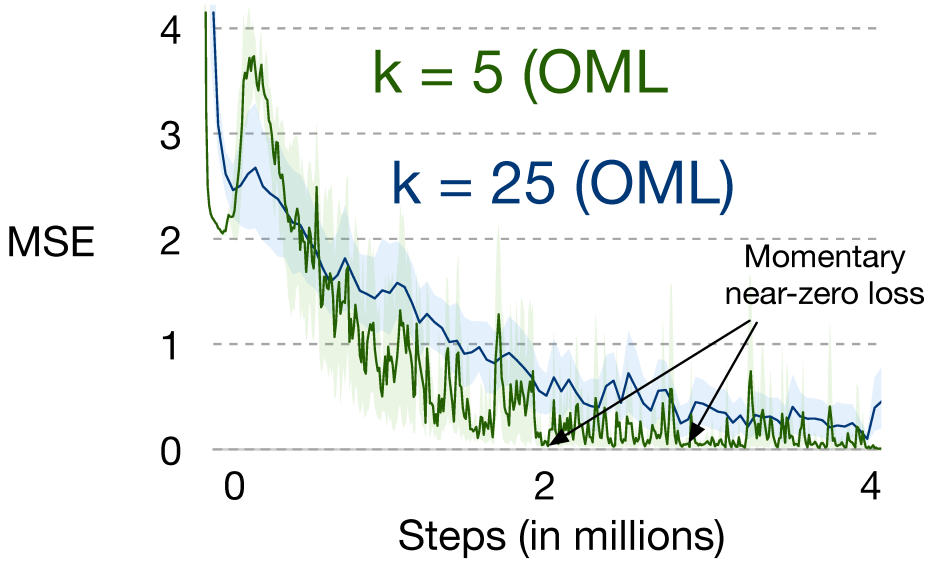


Figure 5.2: Average error on all states as the agent tracks in the environment using a BPTT truncation window of 5 and 25. Both curves are a running average computed using a decay rate of 0.995. In both cases, the learner is incapable of converging to a fixed solution, even though it can momentarily find good solutions. This happens because RLN updates computed on correlated data trajectories are biased, and can interfere with each other. Both experiments were repeated 30 times using random seeds, and the error margins represent 95% confidence intervals using 1,000 bootstrapped samples. The $k = 5$ experiment updates RLN five times more frequently than $k = 25$ and can learn faster, albeit with more noisy updates.

5.1.2 Implementation details

I implement OML using the same neural network architecture used in the last chapter. The RLN network has 5 fully-connected layers, each with 400 neurons, whereas the PLN is a linear predictor. I tune hyper-parameters of OML using hyper-parameters described in Table 4.1. In addition to the values in Table 4.1, I tune the initial value of the adapted learning rate of the PLN by searching over the set $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and report results for the best performing configuration for BPTT truncation windows 5 and 25.

5.1.3 Empirical evaluation

I let OML learn for 4 million steps and report the average error on all states at every time step in Figure 5.2. A learner with enough capacity — certainly

the case for the learner here — should converge to a fixed solution to achieve near-zero error. OML, however, does not converge even after 4 million steps for both $k = 5$ and $k = 25$. Interestingly, it does momentarily find solutions that achieve near-zero loss on all states, but is incapable of maintaining those solutions; instead, it overwrites those solutions with the biased gradient-updates that deteriorate performance. The failure of OML is not surprising: the OML update uses BPTT that is known to struggle when assigning credit over many time steps. Moreover, minimizing loss over a few steps is not sufficient in this MP because minimizing loss on state 3 might increase loss on 3.1 and vice-versa. Increasing the truncation window of OML to $k = 25$ does not address the issue either as gradients can vanish over long horizons.

5.1.4 Discussion

The failure case of OML shows that while OML can learn representations that mitigate interference, the OML update itself suffers from interference when learning from a correlated stream of data. Even when OML does find solutions that achieve near-zero loss momentarily, as shown in Figure 5.2, it is unable to identify them as good and overwrites them with poor updates. In the next chapter, I propose a preliminary solution method that fixes the limitation of OML by verifying that a representation update is good online before committing to it.

Chapter 6

Representation Learning with Backtracking

OML uses a local learning signal — gradient w.r.t few temporally connected states — as a proxy to the true gradient for minimizing regret. This local signal is often helpful, but can also hurt, as shown in the last chapter where it prevented the learner from converging in a ten-state stationary environment. One solution around this limitation is to better estimate the learning signal using a large number of states. Unfortunately, as discussed earlier, this cannot be done online in a scalable way using existing methods. Gradient propagation through many steps is not feasible (See Section 2.4.1), and experience-replay methods do not scale well (See Section 2.2.1).

Estimating a robust signal for updating the representation might not be possible online, but verifying if a given representation update is useful is straightforward — the learner can track loss accumulated over many time-steps after the update and check if the performance has improved. I call the process of checking if a representation update is useful *verification* [1]. Assuming that a learning algorithm proposes good representation updates at-least sometimes, verification is all that is needed to improve a representation online without interference. The idea of verification is a building-block of representation search.

[1]The term is inspired by the following blog post on the need of equipping our learners with the ability to verify their knowledge: http://incompleteideas.net/IncIdeas/Verification.html

6.1 Representation Search

A search algorithm treats representation learning as a search problem. A typical representation search algorithm proposes a solution and verifies if it is effective. If the solution is effective, the algorithm keeps it. Otherwise, it proposes a different solution. Mahmood (2017) and Mahmood and Sutton (2013) proposed a method for feature learning through search called *generate and test*.

6.1.1 Generate and test

Mahmood (2017) and Mahmood and Sutton (2013) proposed an algorithm that continually replaces ineffective features with randomly generated features. They identify ineffective features by looking at the magnitude of the weight associated with the feature for predicting a target. Features with the smallest weights are replaced by new features. They showed that their method can effectively improve the quality of features over time. Moreover, it can be combined with gradient-descent based learning to improve over gradient-descent alone.

Their work, while promising, is not directly applicable to learning representations using deep neural networks. It has two main limitations. First, it generates new features randomly which can be slow, especially if a useful feature is a complex function of the input observation. In contrast, deep learning systems slowly improve existing features using feedback from the environment. Second, they assume that new features can be incorporated in the learner and old features discarded independently of each other. The independence assumption does not hold in the hierarchical features learned by deep neural networks. In a hierarchical representation — believed to be the primary reason behind the impressive empirical successes of neural networks — removing or adding a feature in an arbitrary location can change many other features. Nonetheless, the motivation behind *generate and test* is sound: we need scalable algorithms for learning representations.

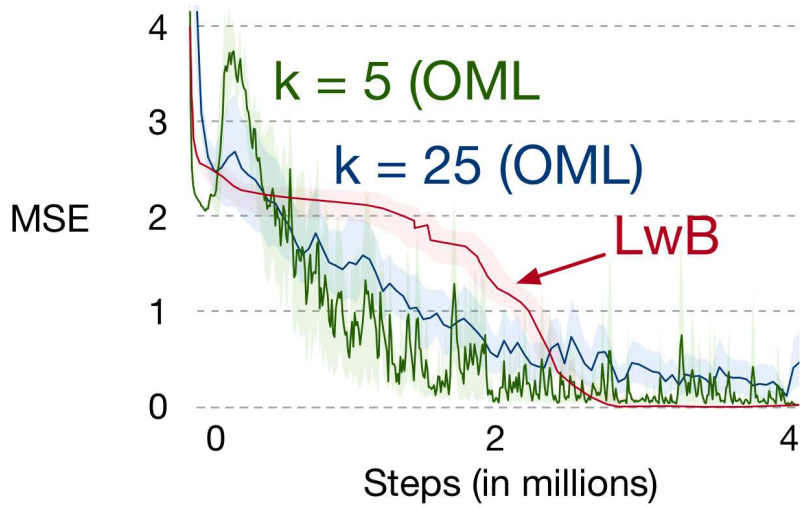


Figure 6.1: Running average of the error on all states of the environment. The OML results are taken from the previous chapter. LwB uses proposals generated using OML with $k = 5$. The proposals are generated using an RLN learning rate sampled uniformly from $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$. Each proposal is verified for 50 steps before accepting or rejecting it. LwB prevents harmful updates, and can converge more robustly to achieve near zero loss. Due to the 50 interactions needed for verification, LwB uses nearly 90% of the interactions with environment to verify proposals. As a result, it learns using an order of magnitude fewer learning updates. All results are averaged over 30 runs, and the confidence intervals represent 95% intervals constructed using 1,000 bootstraps.

6.1.2 Learning with backtracking

I propose Learning with Backtracking (LwB): a representation learning paradigm that can apply search to deep neural networks. LwB does not assume that features are independent of each other and can slowly improve hierarchical feature representations using feedback from the environment. To achieve this, LwB uses a global metric — regret over time — for verifying a representation update. Generate-and-test, in contrast, relies on a feature-level verification metric that only makes sense for independent features.

Being a search-based method, LwB has two components: a proposal generator and a verifier. The proposal generator in LwB proposes representation updates. These updates can come from any method, including local-random search, gradient-based updates, weight pruning, architecture changes *etc*. Given a proposal, the verifier is responsible for deciding if a proposal

should be accepted or rejected. LwB verifies an update by making a copy of the learning network and updating one of the copies with the proposed update. The two networks — called the post-update network and the pre-update network — estimate the regret online for m steps separately. This can be done online for arbitrarily large m without any scalability challenges. Finally, LwB compares the regret achieved by the two networks. If the post-update network achieves lower regret, the proposed representation update is accepted. Otherwise, the learner backtracks to the pre-update network.

The LwB, as described above, can be applied to an online prediction problem. If, however, the sequence of states depends on the actions that a learner takes, such as in control, it would not be possible to simultaneously measure the performance of two networks, the post, and pre-update network. Fixing this limitation is future work, but can be done by comparing the performance of the post-update network to a baseline performance metric, such as a running estimate of the average reward.

6.1.3 Empirical evaluation of LwB

I evaluate LwB on the ten-state MP introduced in the previous chapter using the same 5-layer neural network described in previous chapters. I generate proposals using the OML gradients with $k = 5$ and verify every update proposal for 50 steps before accepting or rejecting it. The learner has access to two models during the verification phase and must select one for making predictions. I use the pre-update RLN for making these predictions as the post-update RLN can result in very high regret if the proposed update is poor. An unintended benefit of backtracking is that I don't have to tune parameters for proposing learning updates for the RLN. The generator can generate proposals by randomly sampling RLN hyper-parameters, and backtracking if the sampled parameters are bad. I implement LwB by uniformly sampling the RLN learning rate from the set $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ for generating proposals.

I report the results in Figure 6.1. LwB can converge to achieve near-zero loss robustly. Once LwB has converged, the verification process assures that future updates do not catastrophically interfere with the learned knowledge.

6.2 Closing Discussion

LwB, as proposed here, is in a preliminary form. I made certain design choices without exploring the alternatives. For instance, I used the biased OML gradients for generating proposals for updating the RLN; OML gradients were sufficient for the simple ten-state benchmark, but might not be sufficient in the general case. A biased gradient-estimate can result in systematically poor proposals. For instance, in my recent work in collaboration with others on learning causal models, I found that gradient-based proposals consistently tried to exploit the salient non-causal features, ignoring the more complex underlying causal features (Javed et al. 2020).

There are two ways LwB can be applied to more interesting problems. First, we can hand-design proposal generators that encode useful inductive biases for the problems we care about. Alternatively, a generator can combine multiple proposal generation methods, and explore their effectiveness online.

Chapter 7

Final Thoughts

My goal in this thesis was to introduce two keys ideas. The first was that we should learn online-aware representations — representations optimized for online-learning. The idea is supported by the observation that a linear transformation of an effective representation can be ineffective for online learning. One-step prediction based representation learning — the current norm in deep-learning research — is incapable of distinguishing between invertible linear transformations of representations, and cannot discover online-aware representations. I introduced OML, one method for learning these representations, but OML relies on BPTT which is not scalable. OML can be combined with *Learning with backtracking* (LwB) to remove some of its limitations.

The second key idea is that while it is hard to compute an accurate estimate of the representation update online, it is possible to verify if an update is useful. LwB builds on this idea. A representation update that improves performance in one part of the state-space can deteriorate performance in a different part. Moreover, two temporally distant states with very different targets can appear similar in the representation space and cause interference. The only way to know that a representation is good is to evaluate the representation on all or at least many of the states the learner cares about. A short-sighted method that confidently updates the representation by looking at a few temporally close states is likely to suffer from interference.

Existing successful methods update their representations using a mini-batch sampled IID from past interactions (Mnih et al. 2015) or by collecting

a large batch of data using parallel actors (Mnih et al. 2016). Both families of method update the representation using information from multiple states, but existing implementations scale poorly with the size of the problem, requiring thousands of samples for a single learning update for larger problems (OpenAI 2018; Vinyals et al. 2019). LwB incorporates information from multiple states while staying scalable.

One problem that I did not adequately address in this thesis is how to generate good proposals for LwB. These proposals can come from gradient-based methods, such as OML (Javed and White 2019); gradient-free random search, such as Perturbations with Backtracking (PwB) (Javed et al. 2020); or even self-supervised and unsupervised methods; it is not clear what is the best approach.

I end the thesis with some predictions. I speculate that a robust online representation learning method would heavily rely on verification of representation updates. It could either use a combination of many proposal generators or learn the proposal generation mechanism online using some form of meta-learning. The meta-learning problem can be formulated by modeling the proposal generation as an optimization problem where the goal is to maximize the probability of having a proposal accepted.

References

Alain, Guillaume, and Yoshua Bengio. 2017. "Understanding intermediate layers using linear classifier probes". *Workshop Track, International conference on learning representations.* 34, 36

Aljundi, Rahaf, et al. 2018. "Memory aware synapses: Learning what (not) to forget". In *European conference on computer vision.* 10

Aljundi, Rahaf, et al. 2019. "Gradient based sample selection for online continual learning". In *Advances in neural information processing systems.* 10

Anand, Ankesh, et al. 2019. "Unsupervised state representation learning in atari". In *Advances in neural information processing systems.* 13, 34, 36

Ans, Bernard, and Stéphane Rousset. 1997. "Avoiding catastrophic forgetting by coupling two reverberating neural networks". *Comptes rendus de l'Académie des sciences-series III-sciences de la Vie.* 10

Bachman, Philip, R Devon Hjelm, and William Buchwalter. 2019. "Learning representations by maximizing mutual information across views". In *Advances in neural information processing systems.* 14

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. "Neural machine translation by jointly learning to align and translate". *International conference on learning representations.* 17

Beaulieu, Shawn, et al. 2020. "Learning to continually learn". *European Conference on Artificial Intelligence.* 37

Bellemare, Marc G, et al. 2013. "The arcade learning environment: An evaluation platform for general agents". *Journal of artificial intelligence research.* 11, 14

Bengio, Y., S. Bengio, and J. Cloutier. 1991. "Learning a synaptic learning rule". In *International Joint Conference on Neural Networks.* 24

Bengio, Yoshua. 2017. "The consciousness prior". *arXiv:1709.08568.* 30

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. "Learning long-term dependencies with gradient descent is difficult". *IEEE transactions on neural networks.* 18

Bengio, Yoshua, et al. 2020. "A meta-transfer objective for learning to disentangle causal mechanisms". *International conference on learning representations.* 15, 23, 25

Brown, Tom B, et al. 2020. "Language models are few-shot learners". *arXiv:2005.14165*. 4, 5, 8

Burgess, Christopher P, et al. 2018. "Understanding disentangling in beta-VAE". *arXiv:1804.03599*. 15

Chaudhry, Arslan, et al. 2019. "Continual learning with tiny episodic memories". *Workshop on multi-task and lifelong reinforcement learning, ICML*. 10

Chen, Tianqi, et al. 2016. "Training deep nets with sublinear memory cost". *arXiv:1604.06174*. 18

Chen, Ting, et al. 2020. "A simple framework for contrastive learning of visual representations". *arXiv:2002.05709*. 14, 34, 36

Cho, Kyunghyun, et al. 2014. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". *arXiv:1406.1078*. 17

Chung, Wesley, et al. 2018. "Two-timescale networks for nonlinear value function approximation". In *International conference on learning representations*. 28

Deng, J., et al. 2009. "ImageNet: A large-scale hierarchical image database". In *Computer vision and pattern recognition*. 8, 12

Doersch, Carl, Abhinav Gupta, and Alexei A Efros. 2015. "Unsupervised visual representation learning by context prediction". In *International conference on computer vision*. 14

Dosovitskiy, Alexey, et al. 2014. "Discriminative unsupervised feature learning with convolutional neural networks". In *Advances in neural information processing systems*. 14

Epstein, Robert. 2016. "The empty brain". *Aeon, May*. 3

Farley, BWAC, and W Clark. 1954. "Simulation of self-organizing systems by digital computer". *Transactions of the IRE Professional Group on Information Theory*. 8

Fedus, William, et al. 2020. "Revisiting fundamentals of experience replay". *arXiv:2007.06700*. 11

Finn, Chelsea, Pieter Abbeel, and Sergey Levine. 2017a. "Model-agnostic meta-learning for fast adaptation of deep networks". *International conference on machine learning*. 24, 30

— . 2017b. "Model-agnostic meta-learning for fast adaptation of deep networks". *International conference on machine learning*. 29

French, Robert M. 1992. "Semi-distributed representations and catastrophic forgetting in connectionist networks". *Connection science*. 9

— . 1997. "Pseudo-recurrent connectionist networks: An approach to the'sensitivity-stability'dilemma". *Connection science*. 10

— . 1999. “Catastrophic forgetting in connectionist networks”. *Trends in cognitive sciences*. 9

Ghiassian, Sina, et al. 2020. “Improving performance in reinforcement learning by breaking generalization in neural networks”. *International Conference on Autonomous Agents and Multiagent Systems*. 9

Gidaris, Spyros, Praveer Singh, and Nikos Komodakis. 2018. “Unsupervised representation learning by predicting image rotations”. *arXiv:1803.07728*. 14

Girshick, Ross. 2015. “Fast r-cnn”. In *International conference on computer vision*. 4

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. 2011. “Deep sparse rectifier neural networks”. In *International Conference on Artificial Intelligence and Statistics*. 30

Goodfellow, Ian, et al. 2014. “Generative adversarial nets”. In *Advances in neural information processing systems*. 8, 13

Gruslys, Audrunas, et al. 2016. “Memory-efficient backpropagation through time”. In *Advances in neural information processing systems*. 18

He, Kaiming, et al. 2015. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In *International conference on computer vision*. 30

— . 2016. “Identity mappings in deep residual networks”. In *European conference on computer vision*. Springer. 8

Heaven, Will Douglas. 2020. “Our weird behavior during the pandemic is messing with AI models”. *MIT Technology Review*, no. May. 2

Hebb, Donald Olding. 1949. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall. 7

Hénaff, Olivier J, et al. 2019. “Data-efficient image recognition with contrastive predictive coding”. *arXiv:1905.09272*. 14

Higgins, Irina, et al. 2017. “beta-VAE: Learning basic visual concepts with a constrained variational framework”. *International conference on learning representations*. 15

Hinton, Geoffrey E, and Ruslan R Salakhutdinov. 2006. “Reducing the dimensionality of data with neural networks”. *science*. 13

Hjelm, R Devon, et al. 2019. “Learning deep representations by mutual information estimation and maximization”. *International conference on learning representations*. 14

Hochreiter, Sepp, et al. 2001. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 39

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long short-term memory”. *Neural computation*. 8, 17

Javed, Khurram, and Faisal Shafait. 2018. "Revisiting distillation and incremental classifier learning". In *Asian conference on computer vision*. 10

Javed, Khurram, and Martha White. 2019. "Meta-learning representations for continual learning". In *Advances in neural information processing systems*. iii, 5, 27, 37, 49

Javed, Khurram, Martha White, and Yoshua Bengio. 2020. "Learning causal models online". *arXiv:2006.07461*. iii, 47, 49

Johnson, Khari. 2020. *OpenAI's Jeff Clune on deep learning's Achilles' heel and a faster path to AGI*. https://venturebeat.com/2020/02/25/openais-jeff-clune-on-deep-learnings-achilles-heel-and-a-faster-path-to-agi/. Accessed: 2020-08-26. 37

Kapturowski, Steven, et al. 2018. "Recurrent experience replay in distributed reinforcement learning". In *International conference on learning representations*. 17

Karpathy, Andrej. 2020. "AI for Full-Self Driving". Youtube. Accessed: 2020-08-26. https://www.youtube.com/watch?v=hx7BXih7zx8. 3

Ke, Nan Rosemary, et al. 2018. "Sparse attentive backtracking: Temporal credit assignment through reminding". In *Advances in neural information processing systems*. 18

Kemker, Ronald, and Christopher Kanan. 2017. "Fearnet: Brain-inspired model for incremental learning". *International conference on learning representations*. 11

Kerg, Giancarlo, et al. 2020. "Untangling tradeoffs between recurrence and self-attention in neural networks". *arXiv:2006.09471*. 19

Kingma, Diederik P, and Jimmy Ba. 2015. "Adam: A method for stochastic optimization". *International conference on learning representations*. 28, 31

Kingma, Diederik P, and Max Welling. 2013. "Auto-encoding variational bayes". *arXiv:1312.6114*. 13

Kirkpatrick, James, et al. 2017. "Overcoming catastrophic forgetting in neural networks". *National academy of sciences*. 10

Kramer, Mark A. 1991. "Nonlinear principal component analysis using autoassociative neural networks". *AIChE journal*. 13

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "Imagenet classification with deep convolutional neural networks". In *Advances in neural information processing systems*. 4, 12

Lecun, Yann. 1985. "Une procedure d'apprentissage pour reseau a seuil asymmetrique". In *Cognitiva 85, Paris, France*. 8

LeCun, Yann, et al. 1998. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE*. 8

Lee, Sang-Woo, et al. 2017. "Overcoming catastrophic forgetting by incremental moment matching". In *Advances in neural information processing systems*. 10

Li, Zhenguo, et al. 2017. "Meta-sgd: Learning to learn quickly for few-shot learning". *arXiv:1707.09835*. 24

Lin, Long-Ji. 1992. "Self-improving reactive agents based on reinforcement learning, planning and teaching". *Machine learning*. 10

Liu, Vincent, et al. 2019. "The utility of sparse representations for control in reinforcement learning". In *AAAI Conference on Artificial Intelligence*. 9

Locatello, Francesco, et al. 2019. "Challenging common assumptions in the unsupervised learning of disentangled representations". In *International conference on machine learning*. 15

Lopez-Paz, David, and Marc'Aurelio Ranzato. 2017. "Gradient episodic memory for continual learning". In *Advances in neural information processing systems*. 10

Machado, Marlos C, et al. 2018. "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents". *Journal of artificial intelligence research*. 11

Mahmood, Ashique. 2017. "Incremental off-policy reinforcement learning algorithms". 16, 44

Mahmood, Ashique Rupam, and Richard S Sutton. 2013. "Representation search through generate and test". In *Workshops track, AAAI Conference on artificial intelligence*. 16, 44

Mathieu, Michael F, et al. 2016. "Disentangling factors of variation in deep representation using adversarial training". In *Advances in neural information processing systems*. 15

McClelland, James L, Bruce L McNaughton, and Randall C O'Reilly. 1995. "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." *Psychological review*. 10

McCloskey, Michael, and Neal J Cohen. 1989. "Catastrophic interference in connectionist networks: The sequential learning problem". In *Psychology of learning and motivation*. Elsevier. 9

Minsky, Marvin. 1961. "Steps toward artificial intelligence". *Proceedings of the IRE*. 16

Minsky, Marvin, and Seymour A Papert. 1969. *Perceptrons: An introduction to computational geometry*. MIT press. 8

Mnih, Volodymyr, et al. 2015. "Human-level control through deep reinforcement learning". *nature*. 4, 10, 11, 28, 48

Mnih, Volodymyr, et al. 2016. "Asynchronous methods for deep reinforcement learning". In *International conference on machine learning*. 11, 49

Moravk, Matej, et al. 2017. "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker". *Science*. 8

Mozer, Michael C. 1989. "A focused back-propagation algorithm for temporal pattern recognition". *Complex systems*. 17

Oord, Aaron van den, Yazhe Li, and Oriol Vinyals. 2018. "Representation learning with contrastive predictive coding". *arXiv:1807.03748*. 14

OpenAI. 2018. "OpenAI Five". 4, 5, 8, 49

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. "On the difficulty of training recurrent neural networks". In *International conference on machine learning*. 18

Rebuffi, Sylvestre-Alvise, et al. 2017. "icarl: Incremental classifier and representation learning". In *Computer vision and pattern recognition*. 10

Redmon, Joseph, et al. 2016. "You only look once: Unified, real-time object detection". In *Computer vision and pattern recognition*. 4

Ren, Shaoqing, et al. 2015. "Faster r-cnn: Towards real-time object detection with region proposal networks". In *Advances in neural information processing systems*. 4

Riemer, Matthew, et al. 2019. "Learning to learn without forgetting by maximizing transfer and minimizing interference". *International conference on learning representations*. 10

Robins, Anthony. 1995. "Catastrophic forgetting, rehearsal and pseudorehearsal". *Connection Science*. 9

Robinson, AJ, and Frank Fallside. 1987. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, MA. 17

Rosenblatt, Frank. 1958. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*. 8

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. "Learning representations by back-propagating errors". *nature*. 8

Schaul, Tom, et al. 2015. "Prioritized experience replay". *arXiv:1511.05952*. 10

Schraudolph, Nicol N. 1999. "Local gain adaptation in stochastic gradient descent". 25, 26

Shah, Haseeb, Khurram Javed, and Faisal Shafait. 2018. "Distillation techniques for pseudo-rehearsal based incremental learning". *arXiv:1807.02799*. 10

Shin, Hanul, et al. 2017. "Continual learning with deep generative replay". In *Advances in neural information processing systems*. 10

Silver, David, et al. 2017. "Mastering the game of go without human knowledge". *Nature*. 8, 16

Silver, David, Richard S Sutton, and Martin Müller. 2008. "Sample-based learning and search with permanent and transient memories". In *International conference on machine learning*. 1

Sutton, Richard S. 1984. "Temporal credit assignment in reinforcement learning". 16, 17

— . 1988. "Learning to predict by the methods of temporal differences". *Machine learning*. 17, 22, 59

— . 1990. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In *Machine learning proceedings*. Elsevier. 17, 22

— . 1992. "Adapting bias by gradient descent: An incremental version of delta-bar-delta". In *AAAI Conference on Artificial Intelligence*. 24, 25, 32

Sutton, Richard S, and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press. 59

Sutton, Richard S, Anna Koop, and David Silver. 2007. "On the role of tracking in stationary environments". In *International conference on machine learning*. 1

Sutton, Richard S, et al. 2000. "Policy gradient methods for reinforcement learning with function approximation". In *Advances in neural information processing systems*. 11

Sutton, Richard S, et al. 2011. "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction". In *International conference on autonomous agents and multiagent systems*. 22, 59

Tesauro, Gerald. 1995. "Temporal difference learning and TD-Gammon". *Communications of the ACM*. 17

Vaswani, Ashish, et al. 2017. "Attention is all you need". In *Advances in neural information processing systems*. 8, 18

Veeriah, Vivek, Shangtong Zhang, and Richard S Sutton. 2017. "Crossprop: Learning representations by stochastic meta-gradient descent in neural networks". In *Joint european conference on machine learning and knowledge discovery in databases*. 25, 26

Vincent, Pascal, et al. 2008. "Extracting and composing robust features with denoising autoencoders". In *International conference on machine learning*. 13

Vincent, Pascal, et al. 2010. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*. 13

Vinyals, Oriol, et al. 2019. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". *Nature*. 4, 5, 8, 49

Watkins, Christopher JCH, and Peter Dayan. 1992. "Q-learning". *Machine learning.* 11

Watkins, Christopher John Cornish Hellaby. 1989. "Learning from delayed rewards". 11

Werbos, Paul. 1974. "Beyond regression:" new tools for prediction and analysis in the behavioral sciences". *Ph. D. dissertation, Harvard University.* 8

Werbos, Paul J. 1988. "Generalization of backpropagation with application to a recurrent gas market model". *Neural networks.* 17

Williams, Ronald J, and David Zipser. 1995. "Gradient-based learning algorithms for recurrent". *Backpropagation: Theory, architectures, and applications.* 17

Yosinski, Jason, et al. 2014. "How transferable are features in deep neural networks?" In *Advances in neural information processing systems.* 12

Zenke, Friedemann, Ben Poole, and Surya Ganguli. 2017. "Continual learning through synaptic intelligence". *Machine learning research.* 10

Zhang, Lijun, Tianbao Yang, Zhi-Hua Zhou, et al. 2018. "Dynamic regret of strongly adaptive methods". In *International conference on machine learning.* 58

Zhang, Richard, Phillip Isola, and Alexei A Efros. 2016. "Colorful image colorization". In *European conference on computer vision.* 14

Zinkevich, Martin. 2003. "Online convex programming and generalized infinitesimal gradient ascent". In *International conference on machine learning.* 21, 58

Appendix A

Generalizations of the Problem Formulation

A.1 Dynamic regret

In Equation 3.1, I defined the goal of the agent as minimizing accumulated loss over time. The online-learning community has looked at an alternative metric: dynamic regret (Zhang et al. 2018; Zinkevich 2003). The idea behind regret — as opposed to accumulated loss — is to remove the irreducible error to get a more interpretable metric. In a non-stationary world, achieving zero loss is impossible — arbitrary changes in the input or target distribution will lead to some mistakes — and looking at the accumulated loss does not tell us if an algorithm is ineffective at learning, or if the the irreducible error is just high. We can define the notion of an optimal model to capture this irreducible error. The optimal model can be defined in terms of a learning algorithm, and an initialization of the model parameters. Let θ_1^* be the initialization parameters for $\hat{f}$. Moreover, assume $\hat{f}_{\theta_1^*}$ is being updated with the optimal learning algorithm — one that achieves the lowest accumulated loss. Then we can define dynamic regret as:

$$Regret_T = \sum_{t=1}^{T} \mathcal{L}(f_t(S_t), \hat{f}^{\theta_t}(S_t) - \mathcal{L}(f_t(S_t), \hat{f}^{\theta_t^*}(S_t)) \tag{A.1}$$

$$= \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y}_t) - \mathcal{L}(y_t, \hat{f}^{\theta_t^*}(S_t)) \tag{A.2}$$

Ideally, we would want to restrict the learning update of the optimal model to satisfy similar computation and memory constraints as our learning algorithm. .

A.2 POMDP Problem Formulation

The online prediction problem can be defined in terms of a Partially Observable Markov Decision Process (POMDP). A POMDP is defined by $(\mathcal{S}, \mathcal{A}, r, p, e)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, and $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S} = P(S_{t+1} = s_{t+1} | A_t = a_t, S_t = s_t)$ is the transition model of the world. The agent takes an action $A_t \in \mathcal{A}$ at time step t. As a result, the world transitions from S_t to $S_{t+1} \in \mathcal{S}$, returning a reward R_{t+1}. Instead of seeing the state of the POMDP — S_{t+1} — directly, the agent sees an observation $O_{t+1} = e(S_{t+1})$, where $e : \mathcal{S} \to \mathcal{O}$ is an unknown function. e could be invertible — converting the POMDP to an MDP — or non-invertible — requiring a recurrent mechanism for constructing the state.

The agent has its own internal state $S_t^{'}$ at time-step t. When e is invertible, O_t has the same information as S_t. Otherwise, the agent has to construct $S_t^{'}$ from partial observations $O_1, \cdots, O_t$. This can be achieved using a state-update function U (Sutton and Barto 2018) that recursively updates the agent state as $S_t^{'} = U(S_{t-1}^{'}, O_t, A_{t-1})$.

Similar to the MP formulation, the goal of the agent is to track a stationary or non-stationary target function $f_t(S_t, A_t) = y_t$. At time step t, the agent outputs $\hat{y}_t$, an estimate of y_t, and takes an action A_t. As a result, the world transitions to a new state S_{t+1} and the agent receives the target label $f_t(S_t, A_t) = y_t$ from the environment. The agent accumulates loss given by $\mathcal{L}(y_t, \hat{y}_t)$, where $\mathcal{L}$ is a loss function that returns the prediction error and the gradient of the error. The target function f can be a function of current state of the agent, such as a function that returns bootstrapped estimate of the return of a value function (Sutton 1988). This formulation can represent important prediction problems, such as learning a model of the world for planning, online self-supervised learning, or learning General Value Functions (GVFs) (Sutton

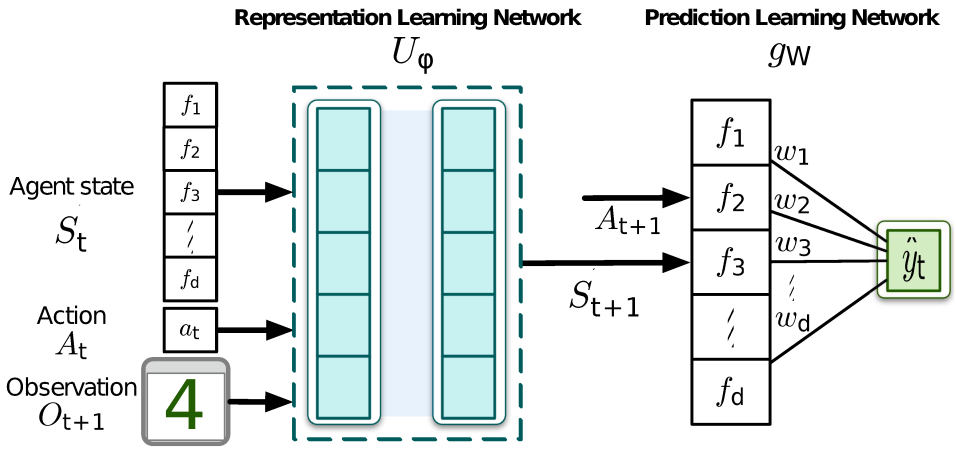


Figure A.1: Architecture for online learning. I decompose the prediction learning function into two components — a Representation Learning Network (RLN) and a Prediction Learning Network (PLN). The RLN represents the state update function U and is recursively updated to incorporated a new observation into the state of the agent. Given U and agent state, the agent learns a prediction learning network (PLN) represented by g_W to make predictions for a target.

et al. 2011).

Given this setting, I define the goal of the agent as minimizing the error accrued over time. Let $\hat{f}^{\theta_t}$ be the agent's estimate of f_t, where θ is a set of tune-able parameters spanning a class of functions. The error up to time step T is defined as:

$$Loss_T = \sum_{t=1}^{T} \mathcal{L}(f_t(S_t, A_t), \hat{f}_{\theta_t}(S_t, A_t)) \tag{A.3}$$

$$= \sum_{t=1}^{T} \mathcal{L}(y_t, \hat{y}_t) \tag{A.4}$$

A.3 POMDP Architecture for Online Learning

The online-learning architecture for POMDP is similar to the MP version, except the RLN represents the state-update function now.

I decompose θ into two functions, g and U, parameterized by W and ϕ respectively such that:

$$\hat{f}_\theta(.) = g_W(U_\phi(S'_{t-1}, O_t, A_{t-1}), A_t). \tag{A.5}$$

Given this decomposition, the agent has to learn two functions for minimizing regret. First, it has to learn the state-update function U_ϕ to recursively update its state. Second, given U_ϕ, the agent has to learn a predictor g_W from (S'_t, A_t) to the target $f^t(S_t, A_t)$. I call U_ϕ the Representation Learning Network (RLN) — it is learning a representation of the state of the world that can be used for making predictions — and g_W the Prediction Learning Network (PLN) as shown in Figure 3.1.